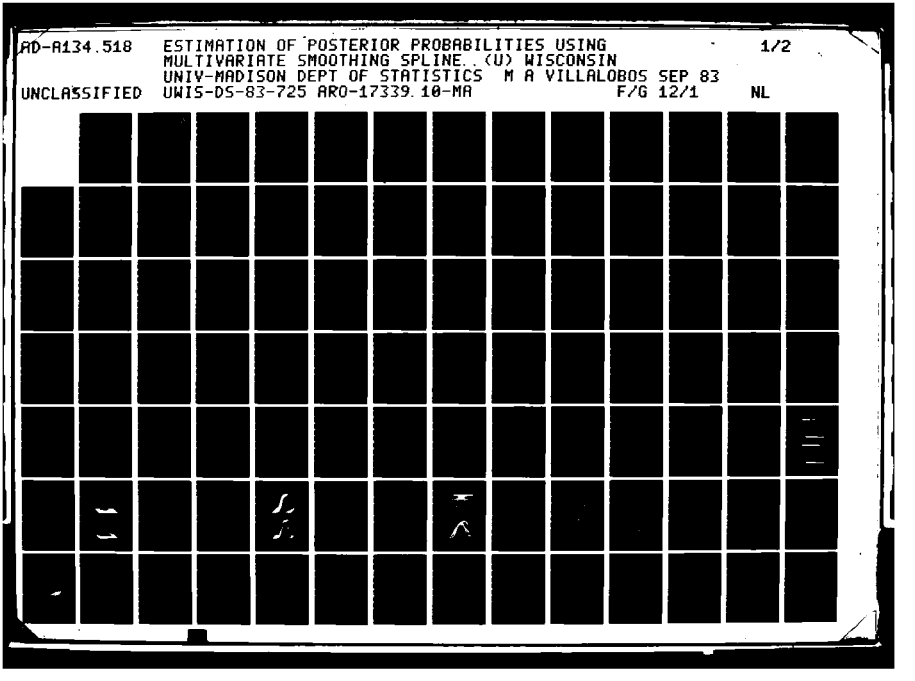
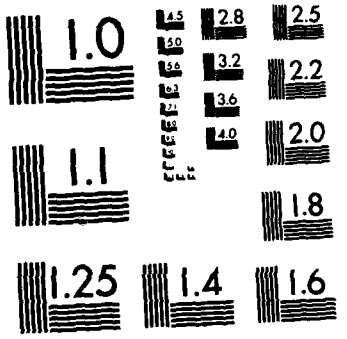
AD-A134 518 ESTIMATION OF POSTERIOR PROBABILITIES USING MULTIVARIATE SMOOTHING SPLINE..(U) WISCONSIN UNIV-MADISON DEPT OF STATISTICS M A VILLALOBOS SEP 83 1/2

UNCLASSIFIED UWIS-DS-83-725 ARO-17339.10-MA F/G 12/1 NL





DEPARTMENT OF STATISTICS
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706

TECHNICAL REPORT NO. 725

September 1983

ESTIMATION OF POSTERIOR PROBABILITIES USING MULTIVARIATE SMOOTHING SPLINES AND GENERALIZED CROSS-VALIDATION

by

Miguel Agustin Villalobos

This research was supported by the Consejo Nacional de Ciencia y Tecnologia-Mexico, by ONR under Contract No. N00014-77-C-0675, and by ARO under Contract No. DAAG29-80-K-0042.

ESTIMATION OF POSTERIOR PROBABILITIES USING MULTIVARIATE SMOOTHING SPLINES AND GENERALIZED CROSS-VALIDATION

Miguel Agustin Villalobos

A thesis under the supervision of Professor Grace Wahba

A nonparametric estimate for the posterior probabilities in the classification problem using multivariate smoothing splines is proposed. This estimate presents a nonparametric alternative to logistic discrimination and to survival curve estimation. It is useful in exploring properties of the data and in presenting them in a way comprehensible to the layman.

The estimate is obtained as the solution to a constrained minimization problem in a reproducing kernel Hilbert space. It is shown that under certain conditions an estimate exists and is unique.

A Monte Carlo study was done to compare the proposed estimate with the two parametric estimates most commonly used. These parametric estimates are based on the assumption that the distributions involved are normal. The spline estimate performed as well as the parametric estimates in most cases where the distributions involved were normal. In the case of non-normal distributions the spline performed consistently better than the parametric estimates.

The computational algorithm developed can be used in the more general context of estimating a smooth function h, when we observe $z_i = L_i h + \varepsilon_i$, $i=1,n$, where ε_i's are independent, zero mean and finite variance random variables, L_i's are linear functionals, and the solution is known a priori to be in some closed, convex set in the Hilbert space, for example, the set of non-negative functions, or the set of monotone functions. This type of problem arises in areas such as cancer research, meteorology and computerized tomography.

We also consider the estimation of the logarithm of the likelihood ratio by a penalized likelihood method. Existence and uniqueness of an estimator under certain conditions is shown. However, a data based method to estimate the "correct" degree of smoothness of the estimator is not given.

Dedicated to

Rosalinda, Miguel Jr. and Karla

To my mother:

Ma. del Refugio Villalobos

To the memory of:

Carmen Bueno
Jose Luis Bueno
Manuel Villalobos

TABLE OF CONTENTS

Abstract ii

Dedication iv

1. Introduction 1
1.1 Scope and background 1
1.2 Conventions and notation 3
1.3 Previous work in nonparametric discriminant analysis 4

2. Spline estimates of posterior probabilities 9
2.1 Motivation 9
2.2 A general minimization problem 11
2.3 The choice of the smoothing parameter 22

3. The algorithm 27
3.1 Introduction 27
3.2 Restatement of the problem 28
3.3 The quadratic programming algorithm 30
3.4 The computation of the approximate GCVC 32
3.5 The step by step algorithm 39

4. Monte Carlo experiments and examples 45
4.1 Comparison of linear, quadratic and spline discrimination 45
4.2 Other simulation results 71
4.3 An example 75

5. Penalized likelihood estimation 78
5.1 Motivation 78
5.2 Existence and uniqueness of an estimator 80

6. Conclusion 88
6.1 Summary 88
6.2 Future work 90

Acknowledgments 90a

Appendix A1: Some mathematical optimization results 91

Appendix A2: Documentation for DSCOMP and DSEVAL 96
DSCOMP 97
DSEVAL 107
Second and third level routines 110

Bibliography 145

CHAPTER 1

INTRODUCTION

1.1 Scope and Background.

Since the early work of Fisher (1936) considerable advances have been made in the practical application of statistical classification techniques. Most of the work in discriminant analysis for continuous variables is based on the assumption that the distributions involved are multivariate Normal, however, in the last ten years, nonparametric methods have received considerable attention, mainly because of the availability of cheaper and faster computers.

In this work we will be concerned with the nonparametric estimation of the posterior probabilities used in Bayesian discriminant analysis. The problem of estimating these posterior probabilities is solved as a particular case of the more general statistical smoothing problem of estimating a smooth function subject to inequality constraints.

Wahba (1979b), introduced the multidimensional smoothing spline in the statistical literature as a tool to model a smooth but otherwise unknown function. She devised the method of Generalized Cross-validation for choosing the parameter that controls the smoothness of the spline based on the data.

Wahba (1980) points out the need for a computational algorithm to solve statistical smoothing problems with linear constraints. This need is also pointed out by Wegman and Wright (1983), who, in the context of isotonic regression say:

> Computational algorithms are clearly the stumbling block in further development of the theory of isotonic splines. When such algorithms become available we believe that smooth, order-preserving non-parametric estimators will sub-

stantially enhance the efficiency of estimation procedures currently in use.

In this thesis we demonstrate the feasibility of doing large multidimensional smoothing problems with inequality constraints. The computational algorithm developed here can be used in applications such as survival curve estimation, logistic regression and the estimation of posterior probabilities. We examine properties of the constrained smoothing spline by a Monte Carlo study.

We believe that the methods presented in this thesis will be useful for exploring properties of the data and presenting them in a way comprehensible to the layman.

In chapter 2 we will present the estimate of the posterior probabilities as the solution to a constrained optimization problem in some suitable space of "smooth" functions. We will describe the method of generalized cross-validation for constrained problems to choose the smoothing parameter.

In chapter 3, we discuss the details of the actual computation of the spline and a step by step algorithm is given.

In chapter 4 some simulation results are presented to compare the spline estimate of the posterior probabilities with the parametric estimates most commonly used. We also present an example with real data.

In chapter 5, the estimation of the logarithm of the likelihood ratio is considered using a Penalized Likelihood approach. The one dimensional results of Silverman (1978) are extended to multiple dimensions and the existence and uniqueness of an estimator under certain conditions is established.

Finally, in chapter 6 we present some concluding remarks and possible directions for future research.

Most of the mathematical optimization background and functional analysis results used throughout this work are presented in appendix A1. The listing of

the documentation for the routines that estimate and evaluate the constrained spline is given in appendix A2.

1.2 Conventions and Notation

Each symbol used is defined at its first occurrence. Vectors are usually denoted by lower case letters and no sub-tildes are used. Matrices are usually denoted by capital letters and are defined by giving their i^{th} row and j^{th} column entry in parenthesis as in the following example:

$$B := (b_{ij}) \quad i=1,\ldots,n;\ j=1,\ldots,m$$

The expression above defines a matrix B as an $n \times m$ matrix with i^{th} row and j^{th} column given by b_{ij}.

The identity matrix of size $n \times n$ is denoted by I_n and O_{nm} denotes an $n \times m$ zero matrix. The subscripts n and m for I and O are dropped when it is clear from the context of the expression what they should be.

The i^{th} element (covariate) of an observations y will be denoted by $y(i)$ and all vectors will be column vectors, for example,

$$y = \begin{bmatrix} y(1) \\ \vdots \\ y(d) \end{bmatrix}.$$

If A is a matrix, then A^t denotes its transpose and A^{-1} denotes its inverse. The trace of a matrix A will be denoted by $tr(A)$. If x is a point in $\mathbf{R}^d$ then $\|x\| = \sqrt{\langle x,x \rangle}$ is the Euclidean norm of x and $\langle x,y \rangle = \sum_{i=1}^{d} x(i)y(i)$.

If h is some function from R to R, then $h^{(k)}$ denotes the k^{th} derivative and when k=1 or 2 we will simply write h' and h''.

If f and g are members of some Hilbert space H, the inner product of f and g is written as $\langle f,g \rangle_H$ and the norm is written as $\|f\|_H$. The subscript H

is dropped when, from the context of the expression, it is clear where the inner product or norm are computed.

Equation (2.3.4) refers to the 4^{th} numbered equation in section 3 of chapter 2. In the text the equation is referred to as (2.3.4). Theorem (5.1.5) refers to the 5^{th} theorem in section 1 of chapter 5 and similarly for Lemmas propositions and definitions.

1.3 Previous work in nonparametric discriminant analysis

Consider k populations $A_1,...,A_k$ and a d-dimensional random vector $X=(X(1),...,X(d))^t$. Assume that the probability distribution of X given that it comes from population A_j , j=1,...,k is absolutely continuous with respect to Lebesgue measure and let $f_j(x)$ denote the corresponding probability density function for j=1,...,k.

Suppose that a training sample $X_{ij}=x_{ij}$, $i=1,...,n_j$, from the population A_j is available for each $j=1,...,k$. Given these training samples and the prior probabilities q_j, $j=1,...,k$ where $0 < q_j < 1$ for $j=1,...,k$ and

$$\sum_{j=1}^{k} q_j = 1$$

we want to estimate the posterior probabilities

$$p_j(x) = q_j f_j(x) / \sum_{i=1}^{k} q_i f_i(x) = P(A_j \mid X=x) \qquad j=1,...,k.$$

The estimates of these posterior probabilities have a clear application in Bayes discriminant analysis.

In this thesis we propose a class of optimization methods for estimating $p_1(x),...,p_k(x)$. For simplicity of notation we will consider the case where we have only two populations since the extension for more than two is straight-

forward.

Most of the work in discriminant analysis (for continuous variables) is based on Normality assumptions, usually with equal covariance matrices. For a summary of the work in discriminant analysis see Lachenbruch and Goldstein (1979). Here we will only be concerned with nonparametric discriminant analysis.

Fix and Hodges (1951) are, to our knowledge, the first to consider the nonparametric classification problem using a k-nearest neighbor approach. For further references related to this paper see Lachenbruch and Goldstein (1979).

During the last 10 years there has been a development of classification rules based on density estimates. These kinds of rules are important because of the extensive research done in nonparametric density estimation. Another feature that makes these kinds of methods attractive is a result by Glick (1972) that says that an estimate of the non-error rate of an arbitrary rule based on parametric or nonparametric density estimators is, in some sense asymptotically optimal provided that:

$$\hat{q}_i \hat{f}_i(x) \xrightarrow{p} q_i f_i(x)$$

pointwise for almost all x in $\mathbf{R}^d$, i=1,...,k, and

$$\int_{\mathbf{R}^d} \sum_{i=1}^{k} \hat{q}_i \hat{f}_i \xrightarrow{p} 1.$$

Kernel, maximum penalized likelihood and orthogonal series density estimates are among the most popular methods. All these density estimation methods involve the choice of a parameter that controls the degree of smoothness of the estimate. Several methods have been proposed to choose the smoothing parameter, among these there are three which are readily computable and objective. Two of these methods were suggested by Wahba (1977 and 1981b) and the third by Habbema, Hermans and Van den Broek (1974). In this

last paper the authors estimate the densities for each population using a kernel estimate. A complete description of kernel methods can be found in Tapia and Thompson (1978).

The kernel estimate used in Habbema, Hermans and Van den Broek (1974) is of the form:

$$\hat{f}_j(x)=(n_j\sigma_j^d)^{-1}\sum_{i=1}^{n_j}K\left\{\frac{(x-x_{ij})}{\sigma_j}\right\} \tag{1.3.1}$$

for j=1,...,k, where, as before d is the dimension of the vector x and k is the number of populations. K is a multivariate normal kernel and the smoothing parameters σ_j are estimated by maximizing what might be called the "cross-validation likelihood function":

$$V(\sigma_j)=\prod_{l=1}^{n_j}\hat{f}_j^{[l]}(x_{jl})$$

where $\hat{f}_j^{[l]}$ is an estimate of f_j computed as in (1.3.1) but leaving out the point x_{jl}. For a detailed description of the algorithm to carry out this kernel discriminant analysis see Hermans and Habbema (1976).

Hermans and Habbema (1975) compare five methods for estimating posterior probabilities using some medical data for which the true posterior probability function is unknown. Four of these five methods are parametric and the fifth one is the kernel method described above. The four parametric methods involve:

(1) Multinormal distributions, equal covariance matrices, estimated parameters.

(2) Multinormal distributions, equal covariance matrices, Bayesian or predictive approach.

(3) Multinormal distributions, unequal covariance matrices, estimated parameters.

(4) Multinormal distributions, unequal covariance matrices, Bayesian or predictive approach.

The nonparametric method is:

(5) Direct estimation of the density functions using a kernel method.

Later, Remme, Habbema and Hermans (1980) carried out a simulation study to compare the performances of methods 1,3 and 5 above. Their simulations show that the performance of the kernel method was either better or as good as the performance of other methods, except in the simulations with multinormal distributions with equal covariance matrices. It performed increasingly well with increasing sample sizes, however, the improvement was very slow in samples simulated from lognormal distributions.

Another nonparametric classification method is given by Chi and Van Ryzin (1977). Their procedure is based upon the idea of a histogram density estimator but bypasses the direct density estimation calculations.

In most of the references listed above the approach has been to estimate each density separately and from this form an estimate of the posterior probabilities. By the Neyman-Pearson lemma, we know that if we want to classify an object as coming from one of two populations with densities f_1 and f_2, we should base the classification on the likelihood ratio f_1/f_2 and hence it would be attractive to have a method to estimate the likelihood ratio directly. Silverman (1978) considers the direct estimation of the log likelihood ratio for one dimensional data. He assumes that $g = log(f_1/f_2)$ is in $C_2(I)$, where I is some interval containing all the observations. He finds the conditional log-likelihood of g and penalizes it according to the smoothness of g using $\int_I (g'')^2$ as the

smoothing penalty functional. He estimates g by maximizing the p.nalized log likelihood and shows that the estimate is a cubic spline. However, he does not give a data-based method to choose the smoothing parameter. In chapter 5 we extend the result in Silverman (1978) to the d-dimensional case.

Van Ness (1980) studied the behavior of the most commonly used discriminant analysis algorithms as the dimension is varied. He found that nonparametric Bayes theorem type algorithms perform better than the parametric (linear and quadratic) algorithms. He also found that the choice of the degree of smoothness must be done with great care or the performance of these nonparametric algorithms can be very poor.

Anderson and Blair (1982) introduce penalized maximum likelihood estimates in the context of logistic regression and discrimination. They obtain estimates of the logistic parameters and a nonparametric spline estimate of the marginal distribution of the regressor x. See also Anderson and Senthilselvan (1980), who use penalty methods for the hazard function in one dimension.

CHAPTER 2

SPLINE ESTIMATES OF POSTERIOR PROBABILITIES

2.1 Motivation

Let $Y_1, \ldots, Y_n$, $n := n_1 + n_2$ denote the combined sample from the two populations A_1 and A_2 and define the random variable

$$Z_i := \begin{cases} 1 \text{ if } Y_i \in A_1 \\ 0 \text{ if } Y_i \in A_2 \end{cases} \tag{2.1.1}$$

Let q_1 and q_2 be the prior probabilities. Our objective is to estimate the posterior probabilities

$$p_j = \frac{q_j f_j}{q_1 f_1 + q_2 f_2} \quad j = 1,2.$$

Since $p_1 + p_2 = 1$, it is enough to obtain an estimate $\hat{p}$ of $p := p_1$ and then the estimate of p_2 is simply $\hat{p}_2 = 1 - \hat{p}$.

In applications the prior probabilities are usually unknown, and hence, instead of estimating p we consider the estimation of

$$h = \frac{w_1 f_1}{w_1 f_1 + w_2 f_2} \tag{2.1.2}$$

where $w_1 = n_1 / n$ and $w_2 = n_2 / n$. Then if $\hat{h}$ is an estimate of h,

$$\hat{p} = \frac{(q_1 / w_1)\hat{h}}{(q_1 / w_1)\hat{h} + (q_2 / w_2)(1 - \hat{h})}$$

is an estimate of p.

We can think of the vector $Z = (Z_1, \ldots, Z_n)^t$ of zeroes and ones as noisy observations on the values $h(y_1), \ldots, h(y_n)$. To see this, note that, if we draw an observation Y from the density f_j with probability w_j, $j = 1,2$, and Z is the

random variable which is 1 or 0 according as j is 1 or 2, then $E(Z \mid Y=y) = h(y)$.

We will assume that all we know about h is that it is a "smooth" function such that $0 \le h \le 1$. For example, in the one dimensional case recall that h'' measures curvature and, hence, we could estimate h by minimizing:

$$Q_\lambda(h) := \frac{1}{n}\sum_{i=1}^{n}(h(y_i)-z_i)^2 + \lambda J_2(h) \qquad (2.1.3)$$

subject to

$$0 \le h(s_i) \le 1 \quad i=1, \ldots, k,$$

where $J_2(h) = \int (h'')^2$, and λ is a parameter that controls the tradeoff between closeness to the data, as measured by the first term in (2.1.3), and smoothness as measured by J_2. The points $s_1, \ldots, s_k$, should form a sufficiently fine mesh so that a smooth function that satisfies the constraints at these points will appear to satisfy them over a set S, where S is such that

$$\sum_{j=1}^{2} w_j f_j > \varepsilon > 0$$

for any $y \in S$, and some given $\varepsilon > 0$.

In section 2 of this chapter we will define the class of functions H where the minimization of (2.1.3) occurs. The function h in H which minimizes (2.1.3) is a piecewise cubic spline (see Schoenberg, 1964).

The generalization of J_2 in two dimensions is

$$J_2(h) = \int_{R^2} \sum_{j=0}^{2} \binom{2}{j} \left[\frac{\partial^2 h(y(1),y(2))}{\partial (y(1))^j \partial (y(2))^{2-j}} \right]^2 dy(1)dy(2).$$

In two dimensions the minimizer of (2.1.3) is called a thin plate spline (Meinguet, 1979) because of the analogy to minimizing the energy of a thin plate of infinite extent. The reader is referred to Wendelberger (1982) for a nice physical interpretation of the piecewise cubic spline in one dimension and the thin plate spline in two.

In more than two dimensions the name thin plate spline does not seem adequate and hence we will use the name Laplacian smoothing splines, suggested by Schoenberg (see Wahba, 1979b), and adopted by Wendelberger (1982).

We can also consider a more general penalty functional than J_2, that is, we will consider a penalty functional J_m to be defined in the following section, which involves the partial derivatives of total order m.

The problem of estimating h is only a particular case of the more general problem of estimating a smooth function f when we have noisy observations on the values that f takes at certain points in $\mathbf{R}^d$, or on the values of some linear functionals $L_1, \ldots, L_n$ applied to f, and we also know that f lies in some closed convex subset of functions.

Following Craven and Wahba (1979), Wahba and Wendelberger (1980) and Wahba (1982) we deal with this more general problem in the following section.

2.2 A general minimization problem

The function space results in this section are given in Meinguet (1978) and Duchon (1976).

Let $H(m,d)$ be the vector space of all Schwartz distributions for which all the partial derivatives in the distributional sense of total order less than m are square integrable over $\mathbf{R}^d$. This definition is given by Meinguet (1978). The space $H(m,d)$ is called a generalized Beppo Levi space of order m over $\mathbf{R}^d$. Adams (1975) calls this space a Sobolev space.

Let $H_0(m,d)$ be the space of polynomials on $\mathbf{R}^d$ of total degree less than m. Then $H_0(m,d)$ is an M-dimensional subspace of $H(m,d)$, where

$$M = \binom{m+d-1}{d},$$

and let $H_1(m,d)$ be the orthogonal complement of $H_0(m,d)$ in $H(m,d)$, that is,

$$H(m,d) = H_0(m,d) \oplus H_1(m,d)$$

where " $\oplus$ " indicates direct sum (see for example Akhiezer and Glazman, 1961).

Let $\tau_M = \{t_1, \ldots, t_M\}$, $t_i \in \mathbb{R}^d$, i=1,...,M, be an M-unisolvent set, that is, a set of M elements of $\mathbb{R}^d$ such that there exists a unique $\varphi \in H_0(m,d)$ with

$$\varphi(t_j) = \beta_j, \ \beta_j \in \mathbb{R}, \ j=1,\ldots,M, \ t_j \in \tau_M.$$

Let $y=(y(1),\ldots,y(d))^t \in \mathbb{R}^d$, $f:\mathbb{R}^d \to \mathbb{R}$, $f \in H(m,d)$. Let $\alpha=(\alpha(1), \ldots, \alpha(d))^t$ and define $|\alpha| := \sum_{j=1}^{d} \alpha(j)$, $\alpha(j) \in \{0,1,\ldots,m\}$. Define the differential operator D^α by:

$$D^\alpha f := \prod_{j=1}^{d} \frac{\partial^{\alpha(j)}}{(\partial y(j))^{\alpha(j)}} f(y) \qquad (2.2.1)$$

If 2m > d, the space $H(m,d)$, equipped with the inner product:

$$\langle f,g \rangle_H = \langle f,g \rangle_0 + \langle f,g \rangle_1$$

where

$$\langle f,g \rangle_0 = \left[\sum_{j=1}^{M} f(t_j) g(t_j) \right], \quad t_j \in \tau_M$$

and

$$\langle f,g \rangle_1 = \sum_{|\alpha|=m} \left[\frac{m!}{\alpha(1)!, \ldots, \alpha(d)!} \right] \langle D^\alpha f, D^\alpha g \rangle_{L_2(\mathbb{R}^d)}$$

can be shown to be a reproducing kernel Hilbert space (RKHS), that is, a Hilbert space where the evaluation functionals are continuous.

Using the results of Duchon (1976) and Meinguet (1979) it can be shown (see Wahba and Wendelberger, 1980 and Wendelberger, 1982), that the reproducing kernel of $H(m,d)$ is given by:

$$K(s,t) = Q(s,t) + P(s,t); \quad s,t \in \mathbb{R}^d \qquad (2.2.2)$$

where

$$Q(s,t) = E_m(s,t) - \sum_{i=1}^{M} p_i(t)E_m(t_i,s) - \sum_{j=1}^{M} p_j(s)E_m(t,t_j) + \sum_{i,j=1}^{M} p_i(s)p_j(t)E_m(t_i,t_j) \tag{2.2.3}$$

and

$$P(s,t) = \sum_{i=1}^{M} p_i(s)p_j(t) . \tag{2.2.4}$$

Here p_j, $j=1,\ldots,M$ are the unique M polynomials in $H_0(m,d)$ satisfying

$$p_i(t_j) = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.2.5}$$

$t_1, \ldots, t_M \in \tau_M$ and $E_m(s,t)$ $s,t \in \mathbb{R}^d$, is given by:

$$E_m(s,t) = \begin{cases} \vartheta_m \|s-t\|^{2m-d} \ln(\|s-t\|) & \text{if } d \text{ is even} \\ \vartheta_m \|s-t\|^{2m-d} & \text{if } d \text{ is odd} \end{cases}$$

where

$$\vartheta_m = \begin{cases} \dfrac{(-1)^{d/2+1+m}}{2^{2m-1}\pi^{d/2}(m-1)!(m-d/2)!}, & d \text{ even} \\ \dfrac{\Gamma(d/2-m)}{2^{2m}\pi^{d/2}(m-1)!}, & d \text{ odd} \end{cases}$$

Let $L_1, \ldots, L_n, N_1, \ldots, N_k$, be continuous linear functionals defined in $H(m,d)$. Suppose that we observe $z_i = L_i f + \varepsilon_i$, i=1,...,n. Where $\varepsilon_1, \ldots, \varepsilon_n$ are independent zero mean random variables with variance covariance matrix given by

$$\sigma^2 D_\sigma^2 := \sigma^2(\delta_{ij}\sigma_i^2), i,j = 1,\ldots,n$$

where $\sigma_i < \infty$, $i=1,\ldots,n$ and

$$\delta_{ij} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases}$$

Here σ^2 is an unknown constant. The σ_i are the relative weights of the measurement errors ε_i. If the variances of ε_i are known to be the same, then we set

$1=\sigma_1=\ldots=\sigma_n$.

We will also assume that we know that the function f is in some closed convex set C that can be well approximated by the set

$$C_k = \left\{ f : N_j f \leq \tau_j, j = 1,\ldots,k \right\}$$

consisting of a finite intersection of half-spaces.

Our objective is to estimate the function f by solving

Problem 2.2.1:

Minimize over H(m,d)

$$\frac{1}{n}\sum_{i=1}^{n}(z_i - L_i f)^2/\sigma_i^2 + \lambda J_m(f) \tag{2.2.6}$$

subject to

$$N_j f \leq \tau_j, \quad j = 1,\ldots,k. \tag{2.2.7}$$

The penalty functional $J_m(f)$ is given by:

$$J_m(f) = \sum_{|\alpha|=m} \left[\frac{m!}{\alpha_1! \cdots \alpha_d!} \right] \|D^\alpha f\|^2_{L_2(\mathbb{R}^d)} \tag{2.2.8}$$

A "constrained Laplacian smoothing spline" is the function $\hat{f}_{n\lambda}$ that solves problem (2.2.1).

Clearly the problem of estimating the function h of section 2.1 is a particular case of problem (2.2.1) when $L_1, \ldots, L_n, N_1, \ldots, N_k$ are evaluation functionals.

Since L_i and N_j are continuous linear functionals, by the Riesz representation theorem (theorem A1.1) there exist functions $\eta_1, \ldots, \eta_n$ and $\xi_1, \ldots, \xi_k$ in the space $H(m,d)$ such that

$$L_i f = \langle \eta_i, f \rangle \quad i = 1,\ldots,n \tag{2.2.9}$$

$$N_j f = \langle \xi_j, f \rangle \quad j = 1,\ldots,n. \tag{2.2.10}$$

The functions η_i and ξ_j are called the representers of L_i and N_j. Any function f

in the Hilbert space $H(m,d)$ can be written as a linear combination of $\eta_1, \ldots, \eta_n, \xi_1, \ldots, \xi_k, p_1, \ldots, p_M$ plus some function ρ which is orthogonal to each η_i, ξ_j, and p_l, that is,

$$f = \sum_{i=1}^{n} c_i \eta_i + \sum_{j=1}^{k} b_i \xi_j + \sum_{l=1}^{M} \tilde{d}_l p_l + \rho \qquad (2.2.11)$$

for some vectors of constants $c=(c_1, \ldots, c_n)^t$, $b=(b_1,\ldots,b_k)^t$ and $\tilde{d}=(\tilde{d}_1,\ldots,\tilde{d}_M)^t$, where

$$<\eta_i,\rho> = <\xi_j,\rho> = <p_l,\rho> = 0,$$

for $i=1,\ldots,n$; $j=1,\ldots,k$ and $l=1,\ldots,M$.

Substituting (2.2.9), (2.2.10) and (2.2.11) in (2.2.6) and (2.2.7) it is easy to show that to solve problem (2.2.1) one should take $\rho=0$, so that, the solution $\hat{f}_{n\lambda}$ can be written as:

$$\hat{f}_{n\lambda}=\sum_{i=1}^{n} c_i \eta_i + \sum_{j=1}^{k} b_i \xi_j + \sum_{l=1}^{M} \tilde{d}_l p_l.$$

By proposition A1.3, η_i and ξ_j are given by:

$$\eta_i(s)=L_{i(t)}K(s,t)$$

$$\xi_j(s)=N_{j(t)}K(s,t).$$

The subscript (t) indicates that the functional L_i (or N_j), is to be applied to what follows considered as a function of t.

Using the Kuhn Tucker theorem we will give a simpler representation for the solution to problem 2.2.1, but first we need to introduce more notation.

First rewrite $\hat{f}_{n\lambda}$ as

$$\hat{f}_{n\lambda} = \begin{bmatrix}\eta \\ \xi\end{bmatrix} a + p^t \tilde{d} \qquad (2.2.12)$$

where $\xi=(\xi_1, \ldots, \xi_n)^t$, $\eta=(\eta_1, \ldots, \eta_k)^t$, $a=(c^t : b^t)^t$ and $p=(p_1, \ldots, p_M)^t$.

Define the matrices:

$$K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}$$

$K_1=[K_{11} : K_{12}]$, $K_2=[K_{21} : K_{22}]$ and $U=[U_1^t : U_2^t]^t$. Where $K_{21}=K_{12}^t$ and the matrices K_{11}, K_{12}, K_{22}, U_1 and U_2 are defined in Table 2.2.1.

TABLE 2.2.1

Matrix	Dimension	(i,j)element	
K_{11}	n x n	$L_{i(s)}L_{j(t)}K(s,t)$	i=1,...,n j=1,...,n
K_{12}	n x L	$L_{i(s)}N_{j(t)}K(s,t)$	i=1,...,n j=1,...,L
K_{22}	L x L	$N_{i(s)}N_{j(t)}K(s,t)$	i=1,...,L j=1,...,L
U_1	n x M	$L_i p_j$	i=1,...,n j=1,...,M
U_2	L x M	$N_i p_j$	i=1,...,L j=1,...,M

We will assume that the following two conditions hold:

Condition 2.2.1

$\eta_1, \ldots, \eta_n, \xi_1, \ldots, \xi_k$ are linearly independent.

Condition 2.2.2

The rank of the matrix U_1 is M.

For example, in the case where $L_i f = f(y_i)$, i=1,...,n and $N_j f = f(s_j)$, j=1,...,k; if the points $y_1, \ldots, y_n, s_1, \ldots, s_k$ are all distinct, condition 2.2.1 will be satisfied. To satisfy condition 2.2.2 we need that $n \geq M$ and that the points $y_1, \ldots, y_n$ uniquely determine an interpolating polynomial of degree m-1.

The following theorem gives a simpler representation for the solution to problem 2.2.1.

2.2.1 Theorem

If conditions 2.2.1 and 2.2.2 hold, the solution to problem 2.2.1 is of the form

$$\hat{f}_{n\lambda}(t) = \sum_{i=1}^{n} c_i L_{i(s)} E_m(s,t) + \sum_{j=1}^{k} b_i N_{j(s)} E_m(s,t) + \sum_{l=1}^{M} d_l \varphi_l(t). \quad (2.2.13)$$

where $\varphi_1, \ldots, \varphi_M$ are the monomial polynomials of degree less than m given by

$$\varphi_l(t) = t(1)^{\mu(1l)} \ldots t(d)^{\mu(dl)}, \quad t \in \mathbb{R}^d \quad (2.2.14)$$

Here $\mu(il) \in \{0,1,\ldots,m-1\}$, $\mu(1l)+\ldots+\mu(dl) < m$, $l=1,\ldots,M$.

Proof

$$L_i \hat{f}_{n\lambda} = L_i[\eta^t : \xi^t]a + L_i p^t \tilde{d}$$

so that

$$\begin{bmatrix} L_1 \hat{f}_{n\lambda} \\ \vdots \\ L_n \hat{f}_{n\lambda} \end{bmatrix} = \left[K_{11} : K_{12}\right]a + U_1 \tilde{d}.$$

Similarly,

$$\begin{bmatrix} N_1 \hat{f}_{n\lambda} \\ \vdots \\ N_k \hat{f}_{n\lambda} \end{bmatrix} = \left[K_{21} : K_{22}\right]a + U_2 \tilde{d}.$$

Let P be a projection operator onto the space $H_1(m,d)$, then:

$$J_m(\hat{f}_{n\lambda}) = \langle P\hat{f}_{n\lambda}, P\hat{f}_{n\lambda} \rangle = \langle [\eta : \xi]a, [\eta : \xi]a \rangle$$
$$= a^t K a.$$

Now we can write problem 2.2.1 as:

Minimize $G(a,d)$ subject to $g(a,d) \leq 0$, where

$$G(a,d) = \frac{1}{2}\left\{[K_{11} : K_{12}]a + U_1\tilde{d} - z\right\}^t D_\sigma^{-2}\left\{[K_{11} : K_{12}]a + U_1\tilde{d} - z\right\} \quad (2.2.14)$$
$$+ \frac{n}{2}\lambda a^t K a$$

$$g(a,d) = [K_{21} : K_{22}]a + U_2\tilde{d} - r \quad (2.2.15)$$

and $D_\sigma^{-2} = diag(1/\sigma_1^2, \ldots, 1/\sigma_n^2)$. The Hessian of the quadratic form $G(a,d)$

is given by:

$$\Xi = \begin{bmatrix} \Xi_{11} & \Xi_{12} & \Xi_{13} \\ \Xi_{21} & \Xi_{22} & \Xi_{23} \\ \Xi_{31} & \Xi_{23} & \Xi_{33} \end{bmatrix}$$

where $\Xi_{21}=\Xi_{12}^t$, $\Xi_{31}=\Xi_{13}^t$, $\Xi_{32}=\Xi_{23}^t$ and:

$$\Xi_{11} = K_{11}D_\sigma^{-2}K_{11}+n\lambda K_{11} \quad \Xi_{12} = K_{11}D_\sigma^{-2}K_{12}+n\lambda K_{12}$$
$$\Xi_{13} = K_{11}D_\sigma^{-2}U_1 \quad \Xi_{22} = K_{21}D_\sigma^{-2}K_{12}+n\lambda K_{22}$$
$$\Xi_{23} = K_{21}D_\sigma^{-2}U_1 \quad \Xi_{33} = U_1^tD_\sigma^{-2}U_1$$

It is easy to see that if conditions 2.2.1 and 2.2.2 hold the Hessian Ξ will be positive definite and hence the solution to problem 2.2.1 exists and is unique.

Now let γ be a $k\times 1$ vector of Lagrange multipliers. By Kuhn-Tucker theorem (Bazaraa and Shetty, 1979, theorem 4.3.6), we have that if $(\hat{a},\tilde{d})$ is the solution to problem 2.2.1, then the following holds:

$$\nabla G(\hat{a},\tilde{d}) + \nabla g(\hat{a},\tilde{d})^t\gamma = 0 \tag{2.2.16}$$

$$\gamma^t g(\hat{a},\tilde{d}) = 0 \tag{2.2.17}$$

$$\gamma \geq 0$$

$$(2.2.17) \Rightarrow g(\hat{a},\tilde{d})^t\gamma\gamma^t g(\hat{a},\tilde{d}) = 0$$

and therefore

$$G(\hat{a},\tilde{d}) = G(\hat{a},\tilde{d})+\frac{1}{2}g(\hat{a},\tilde{d})^t\gamma\gamma^t g(\hat{a},\tilde{d})$$

$$= \frac{1}{2}(K_1\hat{a}+U_1\tilde{d}-z)^t D_\sigma^{-2}(K_1\hat{a}+U_1\tilde{d}-z)+\frac{n\lambda}{2}\hat{a}^t K\hat{a}$$

$$+\frac{1}{2}(K_2\hat{a}+U_2\tilde{d}-r)^t\gamma\gamma^t(K_2\hat{a}+U_2\tilde{d}-r)$$

$$= \frac{1}{2}(K\hat{a}+U\tilde{d}-w)^t S(K\hat{a}+U\tilde{d}-w)+\frac{n\lambda}{2}\hat{a}K\hat{a} \tag{2.2.18}$$

where

$$S := \begin{bmatrix} D_\sigma^{-2} & O_{nk} \\ O_{kn} & \gamma\gamma^t \end{bmatrix}.$$

Also $g(\hat{a},\tilde{d})$ can be written as

$$g(\hat{a},\tilde{d}) = [I_k : O_{kn}][K\hat{a}+U\tilde{d}-w] \tag{2.2.19}$$

where I_k is a $k\times k$ identity matrix and O_{kn} is a $k\times n$ zero matrix. Using (2.2.18) and (2.2.19) the Kuhn-Tucker conditions become:

$$\begin{aligned}\nabla_a G(\hat{a},\tilde{d})+\nabla_a g(\hat{a},\tilde{d})^t\gamma &= KSK\hat{a} + KSU\tilde{d} - KSw \\ &+n\lambda K\hat{a} + Q\begin{bmatrix} I_k \\ O_{nk}\end{bmatrix}\gamma = 0,\end{aligned} \tag{2.2.20}$$

$$\begin{aligned}\nabla_d G(\hat{a},\tilde{d}) + \nabla_d g(\hat{a},\tilde{d})\gamma &= U^t SU\tilde{d} + U^t SK\hat{a} \\ &-U^t Sw + U^t\begin{bmatrix} I_k \\ O_{nk}\end{bmatrix}\gamma = 0\end{aligned} \tag{2.2.21}$$

$$\gamma^t g(\hat{a},\tilde{d}) = \gamma^t[I_k : O_{kn}][K\hat{a}+U\tilde{d}-w] = 0 \tag{2.2.22}$$

and

$$\gamma\geq 0.$$

Now, (2.2.20) implies that

$$K\left\{SK\hat{a}+SU\tilde{d}-Sw+n\lambda\hat{a}+\begin{bmatrix} I_k \\ O_{nk}\end{bmatrix}\gamma\right\}=0. \tag{2.2.23}$$

And (2.2.21) implies

$$U^t\left\{SK\hat{a}+SU\tilde{d}-Sw+\begin{bmatrix} I_k \\ O_{nk}\end{bmatrix}\gamma\right\}=0 \tag{2.2.24}$$

Then, using (2.2.23) and (2.2.24) we get that $-n\lambda U^t\hat{a}=0 \Rightarrow U^t\hat{a}=0$

$$\Rightarrow U_1^t\hat{c} + U_2^t\hat{b} = 0 \tag{2.2.25}$$

and hence

$$\hat{f}_{n\lambda}(t) = \sum_{i=1}^{n}\hat{c}_i L_{i(s)}K(s,t) + \sum_{j=1}^{k}\hat{b}_j N_{j(s)}K(s,t) + \sum_{l=1}^{M}\tilde{d}_l p_l(t) \tag{2.2.26}$$

Substituting (2.2.2), (2.2.3) and (2.2.4) in (2.2.26) and grouping we obtain:

$$\hat{f}_{n\lambda}(t) = \sum_{i=1}^{n} \hat{c}_i L_i(s) E_m(s,t) + \sum_{j=1}^{k} \hat{b}_j N_j(s) E_m(s,t)$$
$$+ \sum_{l=1}^{M} (\tilde{d}_l - d_l^*) p_l(t) - \sum_{l=1}^{M} (U_1^t \hat{c} + U_2^t \hat{b}) E_m(t,t_l)$$
$$+ \sum_{i=1}^{M} \sum_{j=1}^{M} (U_1^t \hat{c} + U_2^t \hat{b}) E_m(t_i,t_j) p_l(t)$$

where d_l^* is given by

$$d_l^* = \sum_{i=1}^{n} \hat{c}_i L_{i(s)} E_m(t_l,s) = \sum_{j=1}^{k} \hat{b}_j N_{j(s)} E_m(t_l,s)$$
$$+ \sum_{i=1}^{n} \hat{c}_i L_{i(s)} p_l(s) + \sum_{j=1}^{k} \hat{b}_j N_{j(s)} p_l(s). \tag{2.2.27}$$

Then by (2.2.25) we have that

$$\hat{f}_{n\lambda}(t) = \sum_{i=1}^{n} \hat{c}_i L_{i(s)} E_m(s,t) + \sum_{j=1}^{k} b_j N_{j(s)} E_m(s,t) \sum_{l=1}^{M} \bar{d}_l p_l(t)$$

where $\bar{d}_l = \hat{d}_l - d_l^*$. Let $\varphi_1, \ldots, \varphi_M$ be as in (2.2.14). Then $\{\varphi_l\}_{l=1}^{M}$ form a basis for $H_0(m,d)$, then, since $p_l = \sum_{j=1}^{M} \vartheta_{lj} \varphi_j$ for some $\vartheta_{lj}{}_{j=1}^{M}$ so that we can substitute the basis $\{p_l\}_{l=1}^{M}$ by the numerically more convenient basis $\{\varphi_l\}_{l=1}^{M}$. That is, we can write:

$$\sum_{l=1}^{M} \bar{d}_l p_l = \sum_{l=1}^{M} \hat{d}_l \varphi_l$$

for some $(\hat{d}_1, \ldots, \hat{d}_M)^t \in \mathbb{R}^d$. Therefore, the solution to problem 2.2.1 can be written in the form (2.2.13).

■

Now let us define the matrix

$$T = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}, \tag{2.2.28}$$

where T_1 and T_2 are given in Table 2.2.2.

Upon replacing the matrices U_1, U_2 and U by T_1, T_2 and T respectively, expressions (2.2.16) through (2.2.25) still hold with $\hat{d}$ instead of $\tilde{d}$ and the condi-

TABLE 2.2.2

Matrix	Dimension	(i,j)element	
E_{11}	n x n	$L_{i(s)}L_{j(t)}E_m(s,t)$	i=1,...,n j=1,...,n
E_{12}	n x L	$L_{i(s)}N_{j(t)}E_m(s,t)$	i=1,...,n j=1,...,L
E_{22}	L x L	$N_{i(s)}N_{j(t)}E_m(s,t)$	i=1,...,L j=1,...,L
T_1	n x M	$L_i\varphi_j$	i=1,...,n j=1,...,M
T_2	L x M	$N_i\varphi_j$	i=1,...,L j=1,...,M

tion $U^t\hat{a} = 0$ is replaced by the condition $T^t a = 0$. Then we can rewrite problem 2.2.1 as:

Problem 2.2.2

Minimize $G(a,d)$ subject to $g(a,d)\leq 0$ and $T^t a = 0$, where now,

$$G(a,d) = \frac{1}{2}(E_1 a + T_1 d - z)^t D_\sigma^{-2}(E_1 a + T_1 d - z) + \frac{n\lambda}{2} a^t E a \qquad (2.2.29)$$

$$g(a,d) = E_2 a + T_2 d - r \qquad (2.2.30)$$

where:

$$E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix},$$

$E_1 = [E_{11} : E_{12}]$, $E_2 = [E_{21} : E_{22}]$ and $E_{21} = E_{12}^t$, E_{11}, E_{12} and E_{22} are given in Table 2.2.2.

Let $N = n + k$, since the rank of T is M, there exists an $N \times N$ matrix $Q = [Q_1 : Q_2]$ such that

$$\begin{bmatrix} Q_1^t \\ Q_2^t \end{bmatrix} T = \begin{bmatrix} R \\ O \end{bmatrix} \qquad (2.2.32)$$

where R is an $M \times M$ upper triangular matrix, O is an $N-M \times M$ zero matrix and Q_1 and Q_2 are $N \times M$ and $N \times N-M$ matrices. This is known as the Q-R decomposition of T (see Dongarra, Bunch, Moler and Stewart, 1979).

Let e be the $N-M$ dimensional vector such that $a = Q_2 e$, then, by (2.2.32) we have $0 = T^t Q_2 e = T^t a$ so that instead of solving problem (2.2.2) we solve

the quadratic programming problem:

Problem 2.2.3

Minimize $G^*(e,d)$ subject to $g^*(e,d) \leq 0$, where:

$$G^*(e,d)=\frac{1}{2}(E_1Q_2e+T_1d-z)^t D_\sigma^{-2}(E_1Q_2e+T_1d-z)+n\lambda e^t Q_2^t E Q_2 e \qquad (2.2.33)$$

and

$$g^*(e,d) = E_2Q_2e + T_2d-r \qquad (2.2.34)$$

Then, if $(\hat{e}, \hat{d})$ solve Problem 2.2.3, $(\hat{a},\hat{d})$ solve Problem 2.2.2, where $\hat{a}=Q_2\hat{e}$.

For a given value of λ we can solve problem 2.2.3 using any quadratic programming routine. So far, nothing has been said about the choice of the smoothing parameter. In the following section we describe the method that we will use to choose a "good" value of λ from the data.

1.4 The choice of the smoothing parameter

In real life problems the correct value of the smoothing parameter λ is not known. Wahba and Wold (1975), Craven and Wahba (1979) and Golub Heath and Wahba (1979) have suggested the use of generalized cross-validation to estimate λ from the data in the unconstrained case. In the presence of linear constraints, Wahba (1980) and (1982) suggested the use of generalized cross-validation for constrained problems.

Before describing the method of generalized cross-validation for constrained problems, which we will refer to as GCVC, we give a brief review of the method of generalized cross-validation for unconstrained problems which will be referred to as GCV.

Let $\hat{f}_{n\lambda}^{[q]}$ be the minimizer of

$$\frac{1}{n}\sum_{i=1,i\neq q}^{n}(L_i f - z_i)^2 + \lambda J_m(f) \tag{2.3.1}$$

If λ is a good choice, then, on the average, $L_q \hat{f}_{n\lambda}^{[q]} - z_q$ should be small and this is reflected in the ordinary cross-validation function $V_0(\lambda)$ given by:

$$V_0(\lambda) = \frac{1}{n}\sum_{q=1}^{n}(L_q \hat{f}_{n\lambda}^{[q]} - z_q)^2. \tag{2.3.2}$$

Craven and Wahba (1979) and Golub, Heath and Wahba (1979) showed that

$$V_0(\lambda) \equiv \frac{1}{n}\sum_{i=1}^{n}\frac{[L_i \hat{f}_{n\lambda} - z_i]^2}{(1-a_{ii}(\lambda))^2} \tag{2.3.3}$$

where $\hat{f}_{n\lambda}$ is the minimizer of

$$Q_\lambda(f) = \frac{1}{n}\sum_{i=1}^{n}(L_i f - z_i)^2/\sigma_i^2 + \lambda J_m(f) \tag{2.3.4}$$

and $a_{ii}(\lambda)$ is the (i,i) entry of the $n \times n$ matrix $A(\lambda)$ satisfying

$$\begin{bmatrix} L_1 \hat{f}_{n\lambda} \\ \vdots \\ L_n \hat{f}_{n\lambda} \end{bmatrix} = A(\lambda)z. \tag{2.3.5}$$

The minimization of (2.3.2) with respect to λ requires the solution of a linear system of order n+M-1, n times for each different value of λ, whereas the minimization of (2.3.3) requires the solution of one linear system of size n+M to find $\hat{f}_{n\lambda}$ and then one of size n to find $a_{ii}(\lambda)$ for each value of λ.

Craven and Wahba (1979) and Golub, Heath and Wahba (1979) show that from the point of view of minimizing the predictive mean square error given by

$$T(\lambda) = \frac{1}{n}\sum_{i=1}^{n}(L_i \hat{f}_{n\lambda} - L_i f)^2, \tag{2.3.6}$$

$V_0(\lambda)$ should be replaced by the generalized cross-validation function $V(\lambda)$ given by:

$$V(\lambda) = \frac{1}{n}\sum_{i=1}^{n} \frac{(L_i \hat{f}_{n\lambda} - z_i)^2}{(1-a_{ii}(\lambda))^2} w_i^2(\lambda) = \frac{\frac{1}{n}\|(I-A(\lambda))z\|^2}{\left(\frac{1}{n}tr(I-A(\lambda))\right)^2} \qquad (2.3.7)$$

where

$$w_i(\lambda) = \frac{1-a_{ii}(\lambda)}{1-\frac{1}{n}\sum_{j=1}^{n} a_{jj}(\lambda)}.$$

They show that the minimizer of (2.3.7) estimates the minimizer of (2.3.6).

Wendelberger (1981) developed an efficient algorithm to compute the minimizer of (2.3.4) estimating λ by minimizing (2.3.6).

Now let C be any closed and convex set in $H(m,d)$, $\hat{f}_{n\lambda}$ be the minimizer of $Q_\lambda(f)$ in C where Q_λ is given in (2.3.4) and let $\hat{f}^{[q]}_{n\lambda}$ be the minimizer in C of

$$\frac{1}{n}\sum_{i=1;i\neq q}^{n} (L_i f - z_i)^2 + \lambda J_m(f). \qquad (2.3:10)$$

The ordinary cross-validation function $V_0(\lambda)$ is given by

$$V_0(\lambda) = \frac{1}{n}\sum_{q=1}^{n} (L_q \hat{f}^{[q]}_{n\lambda} - z_q)^2.$$

It is obvious that $V_0(\lambda)$ would be prohibitive to compute in most cases.

Wahba (1980) shows that given the data

$$\left[z_1, \ldots, z_{q-1}, L_q \hat{f}^{[q]}_{n\lambda}, z_{q+1}, \ldots, z_n\right]^t$$

the minimizer of $Q_\lambda(f)$ in C is $\hat{f}^{[q]}_{n\lambda}$, that is,

$$f_{n\lambda}[z+\delta_q] = \hat{f}^{[q]}_{n\lambda}[z] \qquad (2.3.11)$$

The notation $\hat{f}_{n\lambda}[z+\delta_q]$ indicates that $\hat{f}_{n\lambda}$ is the minimizer in C of $Q_\lambda(f)$ based on the data vector $z+\delta_q$, where δ_q is given by:

$$\delta_q = (0,\ldots,L_q \hat{f}^{[q]}_{n\lambda}[z] - z_q, \ldots, 0)^t,$$

and $\hat{f}^{[q]}_{n\lambda}[z]$ is the minimizer in C of (2.3.10) based on the data vector z.

Using (2.3.11), Wahba (1981a) shows that ordinary cross-validation can be written as

$$V_0(\lambda) = \frac{1}{n}\sum_{q=1}^{n} \frac{(L_q \hat{f}_{n\lambda} - z_q)^2}{(1-a^*_{qq}(\lambda))^2} \tag{2.3.12}$$

where

$$a^*_{qq}(\lambda) = \frac{L_q \hat{f}_{n\lambda}[z+\delta_q] - L_q \hat{f}_{n\lambda}[z]}{L_q \hat{f}_{n\lambda}[z] - z_q}$$

is what Wahba calls the "differential influence" of z_q when λ is used.

The GCVC function is obtained by replacing a^*_{qq} in (2.3.12) by the average "differential influence", so that the GCVC estimate of λ is obtained by minimizing

$$V^C(\lambda) = \frac{\frac{1}{n}\sum_{i=1}^{n}(L_i f_i - z_i)^2}{(1-\frac{1}{n}\sum_{q=1}^{n} a^*_{ii}(\lambda))^2} \tag{2.3.13}$$

As we mentioned in section 2.2 we will assume that the convex set C can be well approximated by the intersection of a finite number of half spaces:

$$C_k = \left\{ f \,:\, N_i f \le r_i,\ i=1,\ldots,k \right\}. \tag{2.3.14}$$

Then, to evaluate $V^C(\lambda)$ for a single value of λ we need to solve n quadratic programming problems in $n+k-M$ variables. To avoid this, Wahba (1981) suggested using the approximate generalized cross-validation function given by:

$$V^C_{app}(\lambda) = \frac{\frac{1}{n}\sum_{i=1}^{n}(L_i \hat{f}_{n\lambda} - z_i)^2}{(1-\frac{1}{n}\sum_{q=1}^{n} a_{qq}(\lambda))^2} \tag{2.3.15}$$

where

$$a_{qq}(\lambda) = \frac{\partial}{\partial z_q} L_q \hat{f}_{n\lambda}|_z.$$

In the following chapter we discuss the algorithm to compute the minimizer of (2.3.4) in the set C_k, estimating the value of λ by minimizing (2.3.15).

CHAPTER 3

THE ALGORITHM

3.1 Introduction.

The software written as part of this thesis was developed for the case where the functionals $L_1, \ldots, L_n$ and $N_1, \ldots, N_k$ are evaluation functionals. That is, we want to minimize

$$\frac{1}{n}\sum_{i=1}^{n}(f(y_i)-z_i)^2 + \lambda J_m(f)$$

subject to $f(s_i) \leq r_i$, for $i=1,\ldots,k$. However, we present the algorithm in its more general form, where the L_i and N_j are any continuous linear functionals.

Before solving problem (2.2.1) we solve the unconstrained problem estimating the value of λ by generalized cross-validation. The software to solve the unconstrained problem in the case where the L_i are evaluation functionals, was developed originally by Wendelberger (1981) and can be obtained from Madison Academic Computing Center (1981) or from IMSL (1983).

If the solution for the unconstrained problem satisfies all the constraints, then that is also the solution to problem 2.2.1.

If this is not the case, we use the value of λ obtained from the solution of the unconstrained problem, say $\hat{\lambda}_0$ as a starting guess for the "correct" λ for problem 2.2.1. In fact, since the imposition of constraints is in some sense a kind of smoothing, it is natural to expect that an optimal λ for the constrained problem will be smaller than $\hat{\lambda}_0$. Also, intuitively, the optimal λ for the constrained problem, say $\hat{\lambda}$ should not be "too far away" from $\hat{\lambda}_0$.

There are two important parts in the algorithm to compute the solution to problem 2.2.1. One is the solution of a quadratic programming problem of size $n+k-M$ for each value of λ that we consider, and the other is the computation of $V^C_{app}(\lambda)$ given by (2.3.15). These two parts are the most intensive in terms of computational effort and hence it is important to try to make them as efficient as possible.

In section 3.2 we restate problem 2.2.1 to follow more closely the way in which the actual computation is done. In section 3.3 we discuss the quadratic programming routine employed. In section 3.4 the computation of the approximate generalized cross-validation function for constrained problems is discussed and in section 3.5 the step by step computational algorithm is presented.

3.2 Restatement of the problem

For computational convenience suppose that instead of observing z, we observe the vector

$$z_\sigma = \begin{bmatrix} z_1/\sigma_1 \\ \vdots \\ z_n/\sigma_n \end{bmatrix}$$

and define the matrices:

$$\begin{aligned} &T_1^\sigma = D_\sigma^{-1} T_1, \quad T^\sigma = \begin{bmatrix} T_1^\sigma \\ T_2 \end{bmatrix}, \\ &E_{11}^\sigma = D_\sigma^{-1} E_{11} D_\sigma^{-1}, \quad E_{12}^\sigma = D_\sigma^{-1} E_{12}, \\ &E_1^\sigma = \begin{bmatrix} E_{11}^\sigma : E_{12}^\sigma \end{bmatrix}, \quad E_2^\sigma = \begin{bmatrix} E_{21}^\sigma : E_{22} \end{bmatrix} \text{and} \\ &E^\sigma = \begin{bmatrix} E_1^\sigma \\ E_2^\sigma \end{bmatrix}, \end{aligned} \tag{3.2.1}$$

where E_{11}, E_{12}, E_{22}, T_1 and T_2 are given in Table 2.2.2.

Following section 2.2 it is easy to see that problem 2.2.3 is equivalent to

Problem 3.2.1

Minimize

$$G^{\sigma}(a_{\sigma},d) = \frac{1}{2}\left(E_1^{\sigma}a_{\sigma} + T_1^{\sigma}d - z_{\sigma}\right)^t \left(E_1^{\sigma}a_{\sigma} + T_1^{\sigma}d - z_{\sigma}\right) + n\lambda a_{\sigma}E^{\sigma}a_{\sigma}$$

subject to

$$g^{\sigma}(a_{\sigma},d) = E_2^{\sigma}a_{\sigma} + T_2 d - r \leq 0 \text{ and } T^{\sigma t}a_{\sigma} = 0.$$

Let $Q = [Q_1^t : Q_2^t]^t$ and R be the Q-R decomposition of T^{σ}, that is,

$$\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} T^{\sigma} = \begin{bmatrix} R \\ O_{(n+k-M)(M)} \end{bmatrix}. \tag{3.2.2}$$

Let e_{σ} be the $n+k-M$ dimensional vector such that $a_{\sigma} = Q_2 e_{\sigma}$. Then, instead of solving problem 3.2.1 we solve the equivalent

Problem 3.2.2

Minimize

$$\bar{G}^{\sigma}(e_{\sigma},d) = \frac{1}{2}\left(E_1^{\sigma}Q_2 e_{\sigma} + T_1^{\sigma}d - z_{\sigma}\right)^t \left(E_1^{\sigma}Q_2 e_{\sigma} + T_1^{\sigma}d - z_{\sigma}\right) + n\lambda e_{\sigma}^t Q_2^t E^{\sigma} Q_2 e_{\sigma}$$

subject to

$$\bar{g}^{\sigma}(e_{\sigma},d) = E_2^{\sigma}Q_2 e_{\sigma} + T_2 d \leq r. \tag{3.2.3}$$

Then, if $(\hat{e}_{\sigma},\hat{d})$ solve problem 3.2.2, $(\hat{a}_{\sigma},\hat{d})$ solve problem 3.2.1, where

$$\hat{a}_{\sigma} = Q_2\hat{e}_{\sigma} = \begin{bmatrix} \hat{c}_{\sigma} \\ \hat{b} \end{bmatrix},$$

and

$$\begin{bmatrix} \hat{a} \\ \hat{d} \end{bmatrix} = \begin{bmatrix} \hat{c} \\ \hat{b} \\ \hat{d} \end{bmatrix} = \begin{bmatrix} D_{\sigma}^{-1}\hat{c}_{\sigma} \\ \hat{b} \\ \hat{d} \end{bmatrix}$$

is a solution to problem 2.2.3.

Now,

$$\begin{aligned}\bar{G}^{\sigma}(e_{\sigma},d) &= \frac{1}{2}\left[\left[E_1^{\sigma}Q_2 : T_1^{\sigma}\right]\begin{bmatrix}e_{\sigma}\\ d\end{bmatrix} - z_{\sigma}\right]^t\left[\left[E_1^{\sigma}Q_2 : T_1^{\sigma}\right]\begin{bmatrix}e_{\sigma}\\ d\end{bmatrix} - z_{\sigma}\right] \\ &\quad + n\lambda\left[e_{\sigma}^t : d^t\right]\begin{bmatrix}Q_2^tE^{\sigma}Q_2 & O_{(n+k-M)(M)}\\ O_{(M)(n+k-M)} & O_{MM}\end{bmatrix}\begin{bmatrix}e_{\sigma}\\ d\end{bmatrix} \\ &= \frac{1}{2}\left[e_{\sigma}^t : d^t\right]\Xi\begin{bmatrix}e_{\sigma}\\ d\end{bmatrix} - z_{\sigma}^t\left[E_1^{\sigma}Q_2 : T_1\right]\begin{bmatrix}e_{\sigma}\\ d\end{bmatrix}\end{aligned} \tag{3.2.4}$$

where Ξ is the Hessian and is given by:

$$\begin{bmatrix}Q_2^tE_1^{\sigma t}E_1^{\sigma}Q_2 + n\lambda Q_2^tE^{\sigma}Q_2 & Q_2^tE_1^tT_1^{\sigma}\\ T_1^{\sigma t}E_{11}^{\sigma}Q_2 & T_1^{\sigma t}T_1^{\sigma}\end{bmatrix}. \tag{3.2.5}$$

Finally, from (3.2.3) we write the term $\bar{g}^{\sigma}(e_{\sigma},d)$ defining the linear constraints as:

$$\bar{g}^{\sigma}(e_{\sigma},d) = \left[E_2^{\sigma}Q_2 : T_2\right]\begin{bmatrix}e_{\sigma}\\ d\end{bmatrix}.$$

3.3 The quadratic programming algorithm

Let $\hat{f}_{n\lambda}(t)$, given by

$$\hat{f}_{n\lambda}(t) = \sum_{i=1}^{n}\hat{c}_iL_{i(s)}E_m(s,t) + \sum_{j=1}^{k}\hat{b}_jN_{j(s)}E_m(s,t) + \sum_{l=1}^{M}\hat{d}_l\varphi_l(t) \tag{3.3.1}$$

be the solution to problem 2.2.1 for a given value of λ. Suppose that there are l active constraints at the solution that correspond to $N_{v(1)}, \ldots, N_{v(l)}$, where

$$\Upsilon := \{v(1), \ldots, v(l)\} \subset \{1,\ldots,k\}. \tag{3.3.2}$$

If we solve problem (2.2.1) for some other value of λ, say λ', then we will get a possibly different set $\Upsilon' = \{v'(1), \ldots, v'(l)\}$ corresponding to the active constraints at the solution. If λ and λ' are relatively close, it is likely that the sets Υ and Υ' will either be the same or at least they will not be "too different".

It is this feature of our problem that motivated the use of an "active set" algorithm to solve the quadratic programming problem. The algorithm that we used was developed by Gill, Gould, Murray, Saunders and Wright (1982). The idea behind this algorithm is as follows.

If the correct active set of constraints were known a priori, then the solution to problem 2.2.1 would be the solution to a problem with equality constraints. There are several efficient algorithms for solving problems with equality constraints and, in fact, the presence of equality constraints actually reduces the dimensionality in which the optimization occurs. Therefore, it is desirable to apply techniques from the equality constrained case to solve problem 2.2.1. To do this, a subset of the constraints of the original problem, called a "working set" of constraints, is selected to be treated as equality constraints. Obviously, the ideal candidate for the working set would be the correct active set. Since the correct active set is not available, the method includes procedures for testing whether the current working set is the correct one, and altering it, adding or deleting constraints, if not.

In our problem, every time we solve the quadratic programming problem 2.2.4 for a given value of λ, say λ', we obtain a correct active set for that particular λ. By the argument at the beginning of this section, this correct active set will be a good starting guess for the correct active set for some other value of λ close to λ' and therefore once we solved the problem for the first time we can expect very fast convergence with this active set algorithm. In fact, this was what we observed in our Monte Carlo studies. In the cases where the active set did not change from one value of lambda to the following, the quadratic programming routine converged in one iteration. We will discuss this further in chapter 4 where we will present the results of the Monte Carlo study.

The Fortran routine to solve the quadratic programming problem was kindly provided by Nicholas I. M. Gould and is based in method 3 of Gill, Gould, Murray, Saunders and Wright (1982).

3.4 The computation of the approximate GCVC

Let $\hat{f}_{n\lambda}$ given by (3.3.1), be the solution to problem 2.2.3. Let $N_{v(1)}, \ldots, N_{v(l)}$ correspond to the l active constraints at the solution, then $\hat{f}_{n\lambda}$ is also the solution to

Problem 3.4.1

Minimize

$$\frac{1}{2}\sum_{i=1}^{n}(L_i f - z_i)^2/\sigma_i^2 + \frac{n\lambda}{2}J_m(f)$$

subject to

$$N_{v(j)} = \tau_{v(j)}, \quad j = 1, \ldots, l.$$

Here, as we will see later, there is a matrix $A(\lambda)$ such that:

$$\begin{bmatrix} L_1\hat{f}_{n\lambda} \\ \vdots \\ L_n\hat{f}_{n\lambda} \end{bmatrix} = A(\lambda)z \tag{3.4.1}$$

Recall now, that the approximate GCVC function is given by

$$V_{app}^{C}(\lambda) = \frac{\frac{1}{n}\sum_{i=1}^{n}(L_i\hat{f}_{n\lambda} - z_i)^2}{(1-\frac{1}{n}\sum_{q=1}^{n}a_{qq}(\lambda))^2} \tag{3.4.2}$$

where

$$a_{qq}(\lambda) = \frac{\partial}{\partial z_q}L_q\hat{f}_{n\lambda}|_z. \tag{3.4.3}$$

After the quadratic programming problem has been solved, the numerator of (3.4.2) can be computed easily. To find the denominator of $V^{C}_{appp}(\lambda)$ we must compute $\sum_{q=1}^{n} a_{qq}(\lambda)$, which, by (3.4.1) is simply the trace of $A(\lambda)$.

In Theorem 3.4.1, we will give an expression for $A(\lambda)$. Before stating this result, we must introduce some more notation. Let the matrices $E_{22}(\mathrm{T})$, $E^{\sigma}_{12}(\mathrm{T})$ and $T_2(\mathrm{T})$ consist of the rows and columns of the matrices E_{22}, E^{σ}_{12} and T_2, corresponding to the active set of constraints, that is, $E_{22}(\mathrm{T})$ consists of rows and columns $v(1), \ldots, v(l)$ of E_{22}, $E^{\sigma}_{12}(\mathrm{T})$ consists of columns $v(1), \ldots, v(l)$ of E^{σ}_{12}, and $T_2(\mathrm{T})$ consists of rows $v(1), \ldots, v(l)$ of T_2. Also define the matrices:

$$\begin{aligned} T^{\sigma}(\mathrm{T}) &= \begin{bmatrix} T^{\sigma}_1 \\ T_2(\mathrm{T}) \end{bmatrix}, \quad E^{\sigma}_{21}(\mathrm{T}) = E^{\sigma}_{12}(\mathrm{T})^t, \\ E^{\sigma}_1(\mathrm{T}) &= \begin{bmatrix} E^{\sigma}_{11} : E^{\sigma}_{12}(\mathrm{T}) \end{bmatrix}, \quad E^{\sigma}_2(\mathrm{T}) = \begin{bmatrix} E^{\sigma}_{21}(\mathrm{T}) : E_{22}(\mathrm{T}) \end{bmatrix} \end{aligned} \tag{3.3.4}$$

and

$$E^{\sigma}(\mathrm{T}) = \begin{bmatrix} E^{\sigma}_{11} & E^{\sigma}_{12}(\mathrm{T}) \\ E^{\sigma}_{21}(\mathrm{T}) & E_{22}(\mathrm{T}) \end{bmatrix}.$$

Where E^{σ}_{11} and T^{σ}_1 are as defined in (3.2.1). Here we use the notation (T) to emphasize the dependence on the set T of active constraints.

Now we can rewrite problem 3.4.1 as:

Problem 3.4.2

Minimize

$$\begin{aligned} G(a_{\sigma}, d) = \frac{1}{2}&(E^{\sigma}_1(\mathrm{T})a_{\sigma} + T^{\sigma}_1 d - z_{\sigma})^t (E^{\sigma}_1(\mathrm{T})a + T^{\sigma}_1 - z_{\sigma}) \\ &+ n\lambda a^t_{\sigma} E^{\sigma}(\mathrm{T}) a_{\sigma} \end{aligned} \tag{3.4.5}$$

subject to

$$\bar{g}(a_\sigma, d) = E_2^\sigma(\mathrm{T})a_\sigma + T_2(\mathrm{T}) - r(\mathrm{T}) = 0 \tag{3.4.6}$$

where $r(\mathrm{T}) = (r_{v(1)}, \ldots, r_{v(l)})^t$.

Let $Q_1(\mathrm{T}), Q_2(\mathrm{T})$ and $R(\mathrm{T})$ form the Q-R decomposition of $T^\sigma(\mathrm{T})$, that is, they satisfy:

$$\begin{bmatrix} Q_1(\mathrm{T}) \\ Q_2(\mathrm{T}) \end{bmatrix} T^\sigma(\mathrm{T}) = \begin{bmatrix} R(\mathrm{T}) \\ O \end{bmatrix}.$$

Here $R(\mathrm{T})$ is $M \times M$, O is $(n+l-m) \times M$, $Q_1(\mathrm{T})$ is $M \times (n+l)$ and $Q_2(\mathrm{T})$ is $(n+l-M) \times (n+l)$. In the proof of theorem 3.4.1 we will assume that $Q_2(\mathrm{T})^t E^\sigma(\mathrm{T}) Q_2(\mathrm{T})$ is positive definite. This will hold if conditions 2.2.1 and 2.2.2 hold (see Dyn and Wahba, 1982).

3.4.1 Theorem

Let $\hat{f}_{n\lambda}$ be the solution to problem 3.4.1, then there exists a matrix $A(\lambda)$ such that

$$\begin{bmatrix} L_1 \hat{f}_{n\lambda} \\ \vdots \\ L_n \hat{f}_{n\lambda} \end{bmatrix} = A(\lambda) \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} \tag{3.4.5}$$

and it is given by:

$$A(\lambda) = I_n - n\lambda Q_2(\mathrm{T}) \left[Q_2(\mathrm{T})^t (E^\sigma(\mathrm{T}) + n\lambda W) Q_2(\mathrm{T}) \right]^{-1} Q_2(\mathrm{T})$$

where $Q_2(\mathrm{T})$ is obtained from the Q-R decomposition of $T^\sigma(\mathrm{T})$, I_n is the $n \times n$ identity matrix and W is given by:

$$W = \begin{bmatrix} I_n & O_{nl} \\ O_{ln} & O_{ll} \end{bmatrix}.$$

Proof

$(3.4.6) \Rightarrow [\bar{g}(a_\sigma, d)]^t [\bar{g}(a_\sigma, d)] = 0$ and therefore we can write (3.4.5) as:

$$\begin{aligned}\bar{G}(a_\sigma,d) &= \bar{G}(a_\sigma,d) + \frac{1}{2}\bar{g}(a_\sigma,d)^t\bar{g}(a_\sigma,d)\\ &= \frac{1}{2}(E_1^\sigma(\Upsilon)a_\sigma + T_1^\sigma d - z_\sigma)^t(E_1^\sigma(\Upsilon)a_\sigma + T_1^\sigma d - z_\sigma)\\ &+ \frac{1}{2}(E_2^g(\Upsilon)a_\sigma + T_2(\Upsilon)d - r(\Upsilon))^t(E_2^g(\Upsilon)a_\sigma + T_2(\Upsilon)d - r(\Upsilon))\\ &+ n\lambda a_\sigma^t E^\sigma(\Upsilon)a_\sigma \qquad (3.4.8)\\ &= \frac{1}{2}(E^\sigma(\Upsilon)a_\sigma + T^\sigma(\Upsilon)d - w)^t(E^\sigma(\Upsilon)a_\sigma + T^\sigma(\Upsilon)d - w) + n\lambda a_\sigma^t E^\sigma(\Upsilon)a_\sigma\end{aligned}$$

where $w = (z_\sigma^t : r(\Upsilon)^t)^t$. Now, write $\bar{g}(a_\sigma,d)$ as:

$$\left[O_{ln} : I_l\right]\left[E^\sigma(\Upsilon)a_\sigma + T^\sigma(\Upsilon)d - w\right]. \qquad (3.4.9)$$

Then if we let $\gamma = (\gamma_1, \ldots, \gamma_l)^t$ be a vector of Lagrange multipliers, problem 3.4.1 reduces to minimizing

$$\bar{G}(a_\sigma,d) + \gamma^t\left[O_{ln} : I_l\right]\left[E^\sigma(\Upsilon)a_\sigma + T^\sigma(\Upsilon)d - w\right] \qquad (3.4.10)$$

Using (3.4.8) and differentiating (3.4.10) with respect to a_σ, d and γ, and equating to zero we obtain:

$$E^\sigma(\Upsilon)E^\sigma(\Upsilon)a_\sigma + E^\sigma(\Upsilon)T^\sigma(\Upsilon)d - E^\sigma(\Upsilon)w + n\lambda E^\sigma(\Upsilon)a_\sigma + E^\sigma(\Upsilon)\begin{bmatrix}O_{nl}\\ I_l\end{bmatrix}\gamma = 0 \qquad (3.4.11)$$

$$T^\sigma(\Upsilon)^t T^\sigma(\Upsilon)d + T^\sigma(\Upsilon)E^\sigma(\Upsilon)a - T^\sigma(\Upsilon)w + T^\sigma(\Upsilon)^t\begin{bmatrix}O_{nl}\\ I_l\end{bmatrix}\gamma = 0 \qquad (3.4.12)$$

and

$$\left[O_{ln} : I_l\right]\left[E^\sigma(\Upsilon)a + T^\sigma(\Upsilon)d - w\right] = 0. \qquad (3.4.13)$$

Then, (3.4.11) implies that

$$E^\sigma(\Upsilon)\left[(E^\sigma(\Upsilon) + n\lambda I_{n+l})a_\sigma + T^\sigma(\Upsilon)d - w\begin{bmatrix}O_{nl}\\ I_l\end{bmatrix}\gamma\right] = 0. \qquad (3.4.14)$$

Equation (3.3.12) implies that

$$T^\sigma(\Upsilon)^t\left\{T^\sigma(\Upsilon)d + E^\sigma(\Upsilon)a_\sigma - w + \begin{bmatrix}O_{nl}\\ I_l\end{bmatrix}\gamma\right\} = 0. \qquad (3.4.15)$$

Then (3.4.13) and (3.4.15) imply that

$$-n\lambda T^{\sigma}(\mathrm{T})^{t} a_{\sigma} = 0 \implies T^{\sigma}(\mathrm{T}) a_{\sigma} = 0 \tag{3.4.16}$$

and (3.4.13) and (3.4.14) imply

$$\begin{aligned}
&\left[O_{ln} : I_l\right]\left[\begin{bmatrix} O_{ln} \\ I_l \end{bmatrix}\gamma + n\lambda I_{n+l} a_{\sigma}\right] = 0 \\
&\implies \gamma = I_l \gamma = -n\lambda\left[O_{ln} : I_l\right] a_{\sigma} \\
&\implies \begin{bmatrix} O_{nl} \\ I_l \end{bmatrix}\gamma = \begin{bmatrix} O_{nl} \\ I_l \end{bmatrix}\left[O_{ln} : I_l\right](-n\lambda) a_{\sigma} \\
&\qquad = \begin{bmatrix} O_{nn} & O_{nl} \\ O_{ln} & -n\lambda I_l \end{bmatrix} a_{\sigma}.
\end{aligned} \tag{3.4.17}$$

Now, using (3.4.17) and (3.4.14) we get

$$\begin{aligned}
\left[E^{\sigma}(\mathrm{T}) + n\lambda I_{n+l}\right] a_{\sigma} &= -T^{\sigma}(\mathrm{T}) d - \begin{bmatrix} O_{nl} \\ I_l \end{bmatrix}\gamma + w \\
&= -T^{\sigma}(\mathrm{T}) d - \begin{bmatrix} O_{nn} & O_{nl} \\ O_{ln} & -n\lambda I_l \end{bmatrix} a_{\sigma} + w \\
\implies \left[E^{\sigma}(\mathrm{T}) + n\lambda W\right] a_{\sigma} &= -T^{\sigma}(\mathrm{T}) d + w.
\end{aligned} \tag{3.4.18}$$

Let $Q_1(\mathrm{T})$, $Q_2(\mathrm{T})$ and $R(\mathrm{T})$ be given by the Q-R decomposition of $T^{\sigma}(\mathrm{T})$. Let e be the $n+l-M$ dimensional vector such that

$$a_{\sigma} = Q_2(\mathrm{T}) e \tag{3.4.19}$$

Then $T^{\sigma}(\mathrm{T}) Q_2(\mathrm{T}) e = T^{\sigma}(\mathrm{T}) a_{\sigma} = 0$, so that (3.4.16) is satisfied. Premultiplying (3.4.18) by $Q_2(\mathrm{T})^{t}$ we obtain:

$$\begin{aligned}
Q_2(\mathrm{T})^{t}\left(E^{\sigma}(\mathrm{T}) + n\lambda W\right) a_{\sigma} &= -Q_2(\mathrm{T})^{t} T^{\sigma}(\mathrm{T}) d + Q_2(\mathrm{T})^{t} w \\
&= Q_2(\mathrm{T})^{t} w \\
\implies Q_2(\mathrm{T})^{t}\left(E^{\sigma}(\mathrm{T}) + n\lambda W\right) Q_2(\mathrm{T}) e &= Q_2(\mathrm{T}) w \\
\implies e &= \left[Q_2(\mathrm{T})\left(E^{\sigma}(\mathrm{T}) + n\lambda W\right) Q_2(\mathrm{T})\right]^{-1} Q_2(\mathrm{T})^{t} w
\end{aligned}$$

and by (3.4.19)

$$a_{\sigma} = Q_2(\mathrm{T})\left[Q_2(\mathrm{T})^{t}\left(E^{\sigma}(\mathrm{T}) + n\lambda W\right) Q_2(\mathrm{T})\right]^{-1} Q_2(\mathrm{T})^{t} w. \tag{3.4.20}$$

Finally, by definition of $A(\lambda)$ we have

$$A(\lambda)w = \begin{bmatrix} L_1\hat{f}_{n\lambda} \\ \vdots \\ L_n\hat{f}_{n\lambda} \\ N_{v(1)}\hat{f}_{n\lambda} \\ \vdots \\ N_{v(l)}\hat{f}_{n\lambda} \end{bmatrix} = E^{\sigma}(T)a_{\sigma} + T^{\sigma}(T)d$$

Therefore,

$$\begin{aligned} I - A(\lambda)w &= w - E^{\sigma}(T)a_{\sigma} - T^{\sigma}(T)d \\ &= n\lambda W a_{\sigma} \qquad (by\ 3.4.18) \\ &= n\lambda W Q_2(T)\left[Q_2(T)^t (E^{\sigma}(T) + n\lambda W) Q_2(T)\right]^{-1} Q_2(T)w \end{aligned}$$

Hence,

$$I - A(\lambda) = n\lambda W Q_2(T)\left[Q_2(T)^t (E^{\sigma}(T) + n\lambda W) Q_2(T)\right]^{-1} Q_2(T)^t. \qquad (3.4.21)$$

■

Now that we have an expression for $A(\lambda)$ we can compute the denominator of $V^{C}_{app}(\lambda)$ given by

$$\left[1 - \frac{1}{n}\sum_{q=1}^{n} a_{qq}(\lambda)\right]^2 = \left[\frac{1}{n} tr\,(I - A(\lambda))\right]^2.$$

Using (3.4.21) we get:

$$\begin{aligned} tr\,(I - A(\lambda)) &= n\lambda tr\left\{Q_2(T)^t W Q_2(T)\left[Q_2(T)^t E^{\sigma}(T) Q_2(T) + n\lambda Q_2(T)^t W Q_2(T)\right]^{-1}\right\} \\ &= n\lambda tr\left\{\Delta[\Phi + n\lambda\Delta]^{-1}\right\}, \end{aligned} \qquad (3.4.22)$$

where $\Delta = Q_2(T)^t W Q_2(T)$ and $\Phi = Q_2(T)^t E^{\sigma}(T) Q_2(T)$.

If we use (3.3.22) to compute $tr\,(I - A(\lambda))$ for each different value of λ we must solve a linear system of size $n + l - M$, to avoid this, we use the following

3.4.1 Proposition

The trace of the matrix $I-A(\lambda)$ is given by:

$$n\lambda\left\{\sum_{i=1}^{n+l-M}\frac{\rho_i}{1+n\lambda\rho_i}\right\}, \tag{3.4.23}$$

where $\rho_1, \ldots, \rho_{n+l-M}$ are the eigenvalues of the real symmetric generalized eigenvalue problem:

$$\Delta\Gamma_i = \rho_i\Phi\Gamma_i, \quad i=1, \ldots, n+l-M, \tag{3.4.24}$$

where $\Gamma_1, \ldots, \Gamma_{n+l-M}$ are the corresponding eigenvectors.

Proof

$$\begin{aligned} tr(I-A(\lambda)) &= n\lambda tr\left\{\Delta[\Phi(I+n\lambda\Phi^{-1}\Delta)]^{-1}\right\} \\ &= n\lambda tr\left\{\Delta(I+n\lambda\Phi^{-1}\Delta)^{-1}\Phi^{-1}\right\} \\ &= n\lambda tr\left\{\Phi^{-1}\Delta(I+n\lambda\Phi^{-1}\Delta)^{-1}\right\}. \end{aligned}$$

Now, let $\rho_1, \ldots, \rho_{n+l-M}$ be the eigenvalues of the generalized eigenvalue problem (3.4.24), then, $\Phi^{-1}\Delta\Gamma_i = \rho_i\Gamma_i$, $i=1, \ldots, n+l-M$ and hence $\rho_1, \ldots, \rho_{n+l-M}$ are the eigenvalues of the matrix $\Phi^{-1}\Delta$, then if UDU^{-1} is the eigenvalue eigenvector decomposition of $\Phi^{-1}\Delta$, with $D = diag(\rho_1, \ldots, \rho_{n+l-M})$, we have:

$$\begin{aligned} tr(I-A(\lambda)) &= n\lambda tr\left\{UDU^{-1}(U[I+n\lambda D]U^{-1}\right\} \\ &= n\lambda tr(D[I+n\lambda D]^{-1}) \\ &= n\lambda\left\{\sum_{i=1}^{n+l-M}\frac{\rho_i}{1+n\lambda\rho_i}\right\}. \end{aligned}$$

■

Therefore, the denominator of $V_{app}^{C}(\lambda)$ is given by

$$[\frac{1}{n}tr(I-A(\lambda))]^2 = \frac{\lambda^2}{n}\left[\sum_{i=1}^{n+l-M}\frac{\rho_i}{1+n\lambda\rho_i}\right]^2. \qquad (3.4.25)$$

Using (3.4.23) to compute $tr(I-A(\lambda))$ has the advantage that we only need to solve the generalized eigenvalue problem when the set of active constraints changes from one value of λ to the next.

In the following section we present the step by step algorithm to compute the solution to problem 2.2.1 choosing the smoothing parameter λ by generalized cross-validation for constrained problems.

3.5 The step by step algorithm

After the unconstrained problem has been solved we have an estimate of λ, say $\hat{\lambda}_0$. The algorithm to compute the constrained spline uses this value of λ as a starting point to get the estimate $\hat{\lambda}$ that minimizes the approximate GCVC function. If $\hat{\lambda}_0 = \infty$ the algorithm also requires the largest eigenvalue of the matrix $Q_2^t E_{11}^q Q_2$ where Q_2 is obtained from the Q-R decomposition of T_1^q:

$$\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} T_1^q = \begin{bmatrix} R \\ O \end{bmatrix}.$$

This eigenvalue, call it ρ^* is available from the routine that computes the unconstrained spline using Wendelberger's (1981) algorithm (see Madison Academic Computing Center, 1981).

The step by step algorithm is as follows:

(1) Compute $M = \binom{m+d-1}{d}$

(2) Compute ϑ_m

(3) Compute T^q given by (3.2.1).

(4) Obtain Q-R decomposition of T^{σ}.

$$\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} T^{\sigma} = \begin{bmatrix} R \\ O \end{bmatrix}.$$

(5) Compute E^{σ} given by (3.2.1)

(6) Compute the Hessian

$$\Xi = \Xi_1 + \begin{bmatrix} n\lambda\Xi_2 & O_{(M)(n+l-M)} \\ O_{(n+l-M)(M)} & O_{MM} \end{bmatrix}$$

First compute

$$\Xi_1 = \begin{bmatrix} Q_2^t E_1^{\sigma} E_1^{\sigma} Q_2 & Q_2^t E_1^{\sigma t} T_1^{\sigma} \\ T_1^{\sigma} E_{11}^{\sigma} Q_2 & T_1^{\sigma t} T_1^{\sigma} \end{bmatrix},$$

then compute

$$\Xi_2 = Q_2^t E^{\sigma} Q_2$$

(7) Compute Matrix defining linear constraints:

$$QPCONS = [E_2^{\sigma} Q_2 : T_2]$$

(8) Compute matrix defining linear term in the quadratic form $G_{\sigma}(e_{\sigma}, d)$

$$QPLIMA = \left[E_1^{\sigma} Q_2 : T_1^{\sigma}\right]$$

(9) Compute linear term in $G_{\sigma}(e_{\sigma}, d)$

$$QPLIVE = z_{\sigma}^t \left[E_1^{\sigma} Q_2 : T_1^{\sigma}\right]$$

(10) Construct regular grid of values of λ around $\hat{\lambda}_{\sigma}$ in logarithmic units, in increments of 0.1 (see details at the end of the step by step algorithm).

(11) For each value of λ do the following:

(11.1) Solve quadratic programming problem to obtain e_{σ}^{*} and d^{*} using the set of active constraints for the solution for the previous value of λ as initial guess.

(11.2) Compute residual sum of squares

(11.3) Compute denominator of approximate GCVC.

(11.3.1) If the set of active constraints for current value of λ is the same as for previous value of λ go to step (11.3.6), otherwise continue to step (11.3.2).

(11.3.2) Compute Q-R decomposition of $T^{\sigma}(T)$

$$\begin{bmatrix} Q_1(T) \\ Q_2(T) \end{bmatrix} T^{\sigma}(T) = \begin{bmatrix} R(T) \\ O \end{bmatrix}$$

(11.3.3) Get matrix Φ given by (3.4.22).

(11.3.4) Compute Δ given by (3.4.22)

(11.3.5) Solve the generalized eigenvalue problem (3.4.24) to obtain $\rho_1, \ldots, \rho_{n+l-M}$.

(11.3.6) Compute denominator of $V_{app}^{C}(\lambda)$ given by (3.4.25).

(11.4) Compute $V_{app}^{C}(\lambda)$ and save.

(11.5) If $V_{app}^{C}(\lambda)$ is smaller than $V_{app}^{C}(previous\ \lambda)$ then $\hat{e}_{\sigma} = e_{\sigma}^{*}$ and $\hat{d} = d^{*}$

(11.6) Next λ

(12) Compute $\hat{c}_{\sigma}$ and $\hat{b}$

$$\begin{bmatrix} \hat{c}_{\sigma} \\ \hat{b} \end{bmatrix} = Q_2^{t} \hat{e}_{\sigma}$$

(13) Get coefficients of the spline $\hat{c}$ $\hat{b}$ and $\hat{d}$:

$$\begin{bmatrix} \hat{c} \\ \hat{b} \\ \hat{d} \end{bmatrix} = \begin{bmatrix} D_{\sigma}^{-1} \hat{c}_{\sigma} \\ \hat{b} \\ \hat{d} \end{bmatrix}$$

In step (10) of the algorithm, the number of values of λ for which we solve the quadratic programming problem and evaluate $V_{app}^{C}(\lambda)$ is given as an input parameter (see documentation of routine DSCOMP in appendix A2). The user can specify the number of values to the left (n_l) and to the right (n_r) of $\hat{\lambda}_o$. We recommend that n_l be greater than n_r since in all our simulation studies the "minimizer" of $V_{app}^{C}(\lambda)$ was to the left of $\hat{\lambda}_o$.

Most of our simulations were done with $n_l = 15$ and $n_r = 10$. One should be careful in choosing n_l and n_r because if the total number of values of λ considered is too large the computation of the spline could be very expensive. As a rule of thumb, and based only on our simulation study, we would suggest considering between 15 to 20 values to the left and between 6 and 10 to the right.

The grid of values of λ is constructed as follows (in units of logarithm of λ):

If $\hat{\lambda}_o < \infty$ the grid is constructed in equally spaced intervals of size 0.1, that is, the grid consists of the following values: $log(\hat{\lambda}_o)-0.1n_l, \ldots, log(\hat{\lambda}_o)-0.1, log(\hat{\lambda}_o), log(\hat{\lambda}_o)+0.1, \ldots, log(\hat{\lambda}_o)+0.1n_r$.

If $\hat{\lambda}_o = \infty$ then we use the sample size n and the largest eigenvalue ρ^* from the unconstrained problem to determine an upper bound for the values of λ that will be considered. This upper bound, call it λ^* is computed as

$$\lambda^* = 10^3(n+k)\rho^*$$

and the values of λ considered are from largest to smallest: $log(\lambda^*)$, $log(\lambda^*)-2.0$, $log(\lambda^*)-3.0$, $log(\lambda^*)-4.0$, $(log(\lambda^*)-4.0)-0.1$, $(log(\lambda^*)-4.0)-0.2, \ldots, (log(\lambda^*)-4.0)-0.1(n_l-3)$.

In step (11.1) we use the routine QPFC to solve the quadratic programming problem.

In step (11.3.5) we use the EISPACK routines REDUC, TRED1 and TQLRAT (see Boyle, Dongarra, Garbow and Moler, 1977), to solve the generalized eigenvalue problem.

The routines DSCOMP and DSEVAL to compute and evaluate the spline are written in Ratfor in a VAX 11/750 under UNIX operating system. Ratfor is a preprocessor which translates this language into portable Fortran. Both, the Ratfor routines and the Fortran routines are available from this author.

All the computations are done in double precision, and the routines are self-documented. In appendix A2 we list the ratfor source for routines DSCOMP and DSEVAL. Routine DSCOMP is the routine that the user should call to solve problem (2.2.1). We also give the listing of all the routines used by DSCOMP and DSEVAL, except the routines to solve the quadratic programming problem. Routine DSEVAL evaluates the spline computed by DSCOMP at a set of points in $\mathbb{R}^d$. The calling sequence for DSCOMP and DSEVAL as well as explanation of the variables that appear in the calling sequence are listed as comments in the source code.

As we mentioned before, the algorithm is written for the case where $L_1, \ldots, L_n$ and $N_1, \ldots, N_k$ are evaluation functionals, for example, $L_i f = f(y_i)$, $i=1, \ldots, n$ and $N_j f = f(s_j)$, $j=1, \ldots, k$.. It is assumed that the $n+k$ points are different so that the generalized eigenvalue problem in step 11.3.5 can be solved. In the near future we plan to incorporate the handling of replicates in the algorithm. One possible strategy to handle replicates is the following: suppose that we have n_i replicates at the point y_i, and denote them as $z_{i(1)}, \ldots, z_{i(n_i)}$, then take the average

$$\bar{z}_i = \frac{\sum_{j=1}^{n_i} z_{i(j)}}{n_i}$$

and let $\sigma_i^2 = 1/n_i$. Then use $(\bar{z}_i, y_i)$, $i=1, \ldots, n$ as the data with relative

weights $(\sigma_1^2, \ldots, \sigma_n^2)$. The grid of points $s_1, \ldots, s_k$ can always be chosen so that $s_j \neq y_i$, $i=1, \ldots, n$ $j=1, \ldots, k$.

In the example with real data in section 4.2 the replicates were handled using the strategy mentioned above.

CHAPTER 4

MONTE CARLO EXPERIMENTS AND EXAMPLES

4.1 Comparison of linear, quadratic and spline discrimination

In this section we compare the performance of the two parametric models most commonly used with the spline model.

Design of the simulation study

Our study is restricted to continuous bivariate data and to discrimination between two populations A_1 and A_2. The prior probabilities are taken to be equal and so are the sample sizes for the training samples.

As in chapter 2, let $Y_1, \ldots, Y_n$, $n=n_1+n_2$, denote the combined sample from the two populations and define

$$Z_i = \begin{cases} 1 & \text{if } Y_i \in A_1 \\ 0 & \text{if } Y_i \in A_2 \end{cases}.$$

Then, since for the simulation study the priors and the sample sizes n_1 and n_2 are equal, we have that

$$P[Z=1 \mid Y=y] = \frac{f_1(y)}{f_1(y)+f_2(y)} = p(y) = h(y). \qquad (4.1.1)$$

The three models that we will consider are the following:

L- LINEAR: The densities f_1 and f_2 in (4.1.1) are assumed to be bivariate Normal with possibly different means and the same variance covariance matrix Σ. The means are estimated by the sample means and Σ is estimated by the pooled sample variance covariance matrix. This gives rise to linear discriminant analysis which is frequently applied. See for example BMDP (1975).

Q- QUADRATIC: The densities f_1 and f_2 in (4.1.1) are assumed to be bivariate Normals with possibly different mean vectors and variance covariance matrices Σ_1 and Σ_2. The means are estimated by the sample means and Σ_1 and Σ_2 are estimated by the sample variance covariance matrices.

S- SPLINE: The posterior probability $p(y)$ in (4.1.1) is estimated directly by minimizing

$$\frac{1}{n}\sum_{i=1}^{n}(z_i - p(y_i))^2 + \lambda J_2(p)$$

subject to

$$0 \le p(s_j) \le 1, \quad j = 1, \ldots, k,$$

where $J_2(p)$ is given by (2.2.8) with $m=2$ and the smoothing parameter λ is estimated by generalized cross-validation for constrained problems as described in section 2.3.

The spline was computed as follows: First the unconstrained problem was solved and an estimate $f^u_{n\lambda}$ was obtained using Wendelberger's (1981) algorithm. A regular grid of 15×15 points was constructed in the range of the data and the unconstrained spline was evaluated at these 225 points. If the constraints were violated at some of these 225 points, then the constrained problem was solved using $\hat{\lambda}$ from the unconstrained problem as initial value for λ for the constrained problem. The set of points $s_1, \ldots, s_k$ at which the constraints were enforced consisted of the subset of of the 225 points at which either $f^u_{n\lambda} > 0.9$ or $f^u_{n\lambda} < 0.1$. In all our simulations we restricted k to be less than 100.

It would have been desirable to compare the spline model with the Kernel model of Habbema, Hermans and Van den Broek (1974), unfortunately we could not get their software.

In each simulation, training data were generated to estimate the posterior probability (4.1.1) according to each of the three models. We fixed the sample size at $n_1 = n_2 = 70$. Also 100 additional data points were generated from each population to serve as test data.

The IMSL (1982) routines GGNSM and GGUBS were used to simulate the data from four different types of distributions. Using the notation $N_2(\mu_1,\mu_2,\sigma_{11},\sigma_{22})$ for the uncorrelated bivariate normal, the four types of distributions are given in Table 4.1.

Density contours for each type of distribution are given in Figure 4.1.1

For each type of distribution four simulations were done. In each simulation, 140 training and 200 test observations were generated and the values of six measures of performance were calculated for each model. These values were averaged over the four simulations and their standard deviations were computed. For distribution type three six more simulations were done with training samples of size 50 for each group and three simulations with training samples of size 90.

TABLE 4.1.1

Sample	f_1	f_2
1	$N_2(0,0,1,1)$	$N_2(2,0,1,1)$
2	$N_2(0,0,1,1)$	$N_2(5,0,16,16)$
3	$N_2(0,0,1,1)$	$\frac{1}{2}N_2(1.5,-2.5,1,1)+\frac{1}{2}N_2(1.5,2.5,1,1)$
4	$\frac{1}{2}N_2(0,5,1,1)$ $+\frac{1}{2}N_2(0,-5,1,1)$	$N_2(0,0,16,16)$

Density Contours

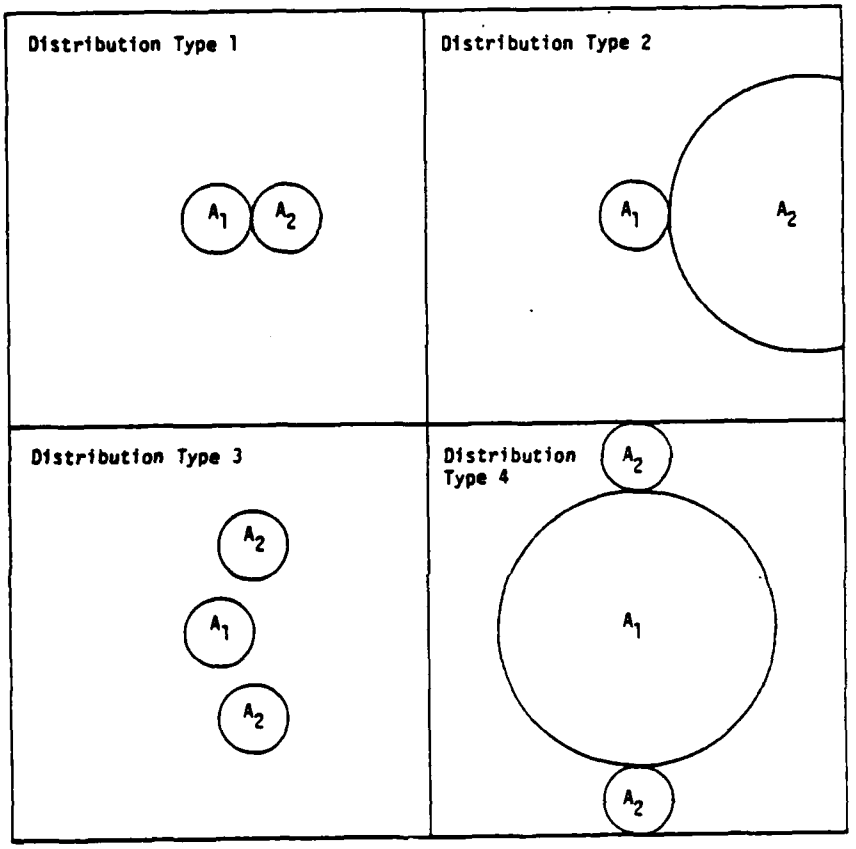


Figure 4.1.1

Measures of Performance

We use misclassification rate as a criteria for measuring how well each of the three models performs. The simplest way of estimating misclassification rate is by the proportion of training samples that are misclassified. However, this leads to an optimistic result, and unless the training sample is perfectly representative of the population, the classifier will reflect peculiarities of the sample at hand that do not exist in the population. The error rate calculated by reclassifying the design set is known as "apparent error rate". The "true error rate" of a classifier is the expected error rate of this classifier on future samples from the same population.

One of the advantages of a simulation study is that we can generate a test sample from the same distribution of the training sample and get a better estimate of the true misclassification rate from the proportion of test elements that were misclassified.

Several authors like Gilbert (1968), Lachenbrook and Mickey (1968), Marks and Dunn (1974), Goldstain (1975), Van Ness and Simpson (1976), Aitchison, Habbema and Kay (1977) and Remme, Habbema and Hermans (1980) have evaluated various discriminant analysis models.

We choose to use the same measures of performance that Remme, Habbema and Hermans (1980) used to compare their Kernel model with the linear and quadratic models. All this performance measures are computed on the test sample.

Two kinds of allocation rules are considered. One is the "forced allocation rule" (FAR) which consists of allocating the test element to the population with higher posterior probability, that is, the test element t is allocated to A_1 if $\hat{p}(t)>0.5$ and to A_2 otherwise. This kind of rule does not take into account differences in the posterior probabilities between for example 0.55 and 0.95.

This lead us to the consideration of a "doubt allocation rule" (DAR). This allocation rule is as follows: allocate test element t to population A_1 if $\hat{p}(t)$ is greater than some prespecified threshold value $\delta>0.5$; allocate t to A_2 if $\hat{p}(t)<1-\delta$ and allocate t to a "doubt category" if $1-\delta \leq \hat{p}(t) \leq \delta$. The threshold value used in our simulations is $\delta=0.9$.

We have two groups of performance measures. The first group is used to compare the models L, Q and S among themselves, that is, without using the knowledge of the true posterior probabilities. These first group of measures are:

P1: Percentage of test elements allocated to population of origin using FAR.

P2: Percentage of test elements allocated to population of origin using DAR.

P3: Percentage of test elements allocated to population from which they did not originate using DAR.

The following three measures compare each estimate of the posterior probability under the three models with the true posterior probability. These measures are more adequate for evaluating the estimation of the posterior probability. These measures are:

P4: Percentage of test elements allocated to the same population with both the true and the estimated posterior probability using (FAR).

P5: Percentage of test elements for which there is "strong agreement" between true and estimated posterior probability using DAR (see bellow).

P6: Percentage of test elements for which there is "strong disagreement" between the true and estimated posterior probabilities using DAR (see bellow).

Measures P5 and P6 are computed as follows. For each test element $p(t)$ and $\hat{p}(t)$ were computed and the test element was allocated to one of the 9

categories in table 4.1.2 according to the values of $p(t)$ and $\hat{p}(t)$. Then, P5 is the computed as the percentage of test elements that fall in classes c_{11}, c_{22} and c_{33}. P6 is the percentage of test elements that fall in classes c_{31} and c_{13}.

TABLE 4.1.2

		p		
		[0.0,0.1]	(0.1,0.9)	[0.9,1.0]
$\hat{p}$	[0.0,0.1]	c_{11}	c_{12}	c_{13}
	(0.1,0.9)	c_{21}	c_{22}	c_{23}
	[0.9,1.0]	c_{31}	c_{32}	c_{33}

Results

For each of the four types of distributions the average values of P1, P2, P3, P4, P5 and P6 over the four simulations are given in tables 4.1.3 through 4.1.6. The column labeled U.S. in these tables corresponds to the unconstrained spline.

This information is summarized in figures 4.1.2 through 4.8.7.

TABLE 4.1.3

Distribution Type 1

Perf. Measure	L	Stan. Dev.	Q	Stan. Dev.	S	Stan. Dev.	U.S.	Stan. Dev.
P1	82.50	2.58	81.38	2.95	81.25	3.40	81.50	2.55
P2	65.13	1.80	65.13	1.38	62.25	3.01	61.00	2.74
P3	4.13	1.25	4.38	1.80	3.50	2.04	2.75	1.19
P4	97.75	1.19	96.88	0.75	95.75	2.78	96.25	2.22
P5	90.63	2.63	88.50	5.74	88.75	7.79	92.25	3.38
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

TABLE 4.1.4

Distribution Type 2

Perf. Measure	L	Stan. Dev.	Q	Stan. Dev.	S	Stan. Dev.	U.S.	Stan. Dev.
P1	86.00	1.08	90.13	2.17	92.38	1.03	92.38	1.03
P2	43.63	5.45	79.38	2.78	70.00	9.60	68.25	8.23
P3	0.75	0.65	3.63	2.63	0.63	0.75	0.50	0.71
P4	88.63	1.44	97.25	1.19	95.50	1.08	96.50	0.82
P5	41.63	10.36	95.88	0.48	76.75	10.90	75.38	11.12
P6	1.75	0.96	0.50	0.71	1.50	0.91	1.50	0.91

TABLE 4.1.5

Distribution Type 3

Perf. Measure	L	Stan. Dev.	Q	Stan. Dev.	S	Stan. Dev.	U.S.	Stan. Dev.
P1	69.25	2.40	84.50	2.12	86.25	3.40	86.00	2.61
P2	36.38	2.46	52.38	3.17	68.13	3.79	64.5	6.18
P3	0.38	0.48	0.50	0.41	1.63	1.11	1.13	0.95
P4	71.63	2.66	90.88	1.80	95.13	1.49	92.13	8.18
P5	41.63	3.57	70.38	3.77	86.38	3.97	73.50	19.67
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.50

TABLE 4.1.6

Distribution Type 4

Perf. Measure	L	Stan. Dev.	Q	Stan. Dev.	S	Stan. Dev.	U.S.	Stan. Dev.
P1	53.25	5.01	88.13	2.39	92.13	2.25	93.63	1.31
P2	28.13	5.12	60.25	5.07	80.63	1.60	81.25	1.50
P3	0.00	0.00	0.50	0.58	2.13	1.49	1.00	1.08
P4	51.50	4.80	91.38	2.63	94.38	0.85	95.63	1.18
P5	35.25	2.50	55.63	6.57	73.25	3.93	77.25	4.66
P6	0.00	0.00	0.00	0.00	0.25	0.50	0.00	0.00

It is clear from this figures that the linear model performs well only with samples from distribution type 1, that is, when both samples come from bivariate normals with the same variance covariance matrix. Even in this case the performance of the quadratic or the spline models is almost as good as the performance of the linear model. When the samples are generated from two bivariate normals with different variance covariance matrices, the quadratic model

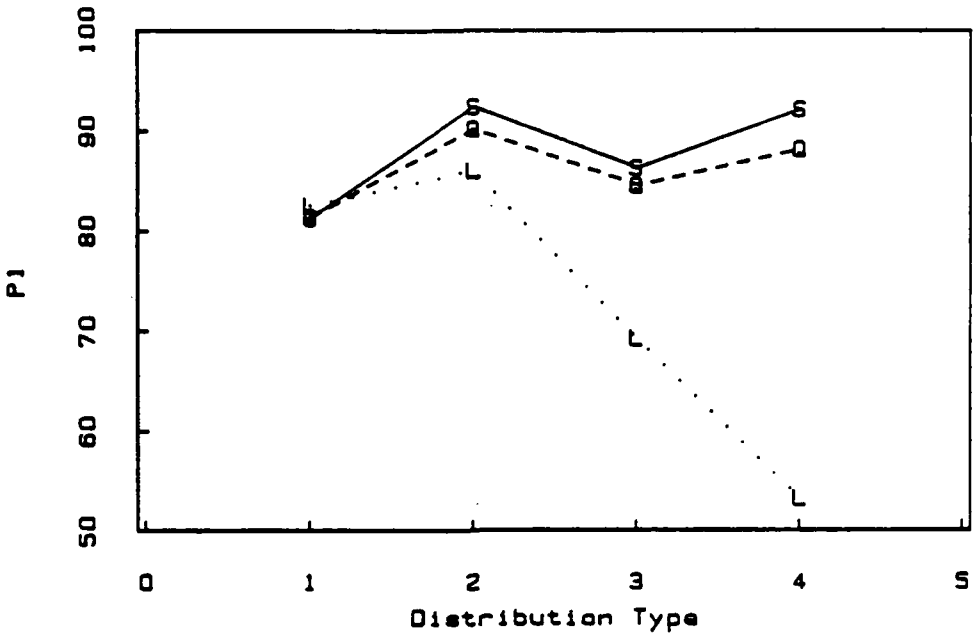


Figure 4.1.2: Percentage of test elements classified correctly using Forced Allocation Rule.
S--- Const. Spline, Q- - Quadratic, L... Linear.

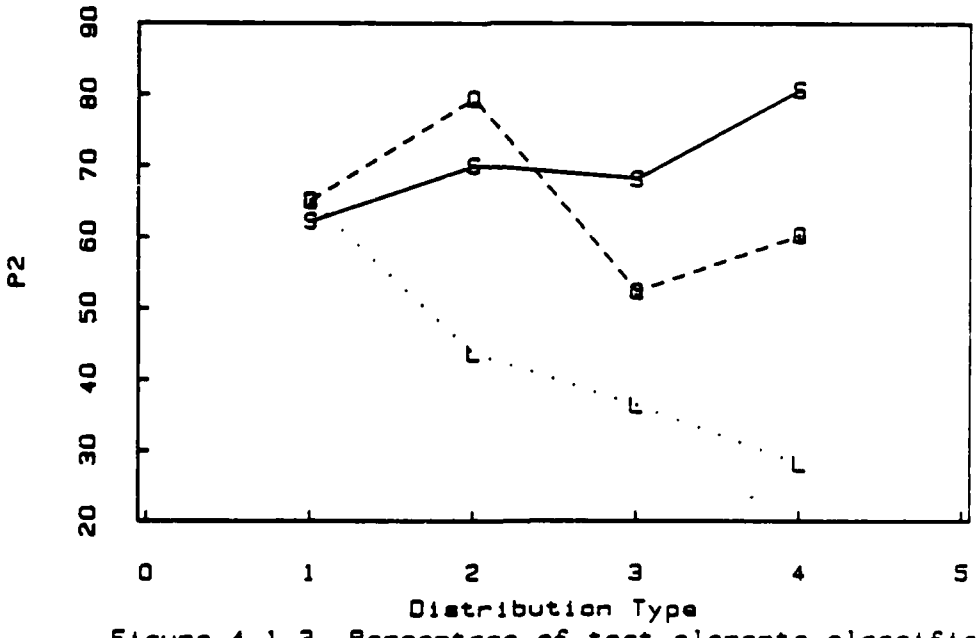


Figure 4.1.3. Percentage of test elements classified correctly using Doubt Allocation Rule.
S--- Const. Spline, Q- - Quadratic, L... Linear.

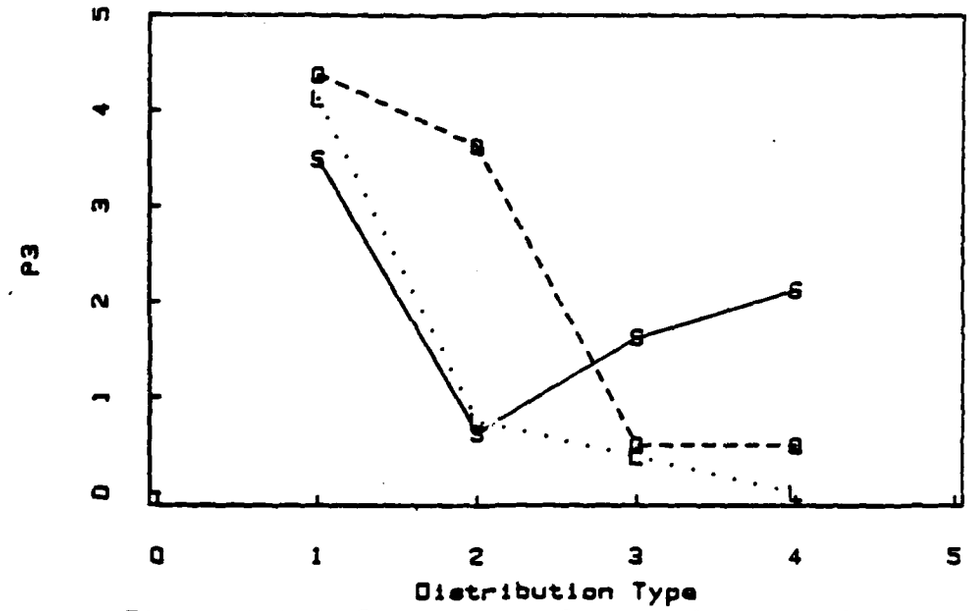


Figure 4.1.4. Percentage of test elements classified incorrectly using Doubt Allocation Rule.
S--- Const. Spline, Q- - Quadratic, L... Linear.

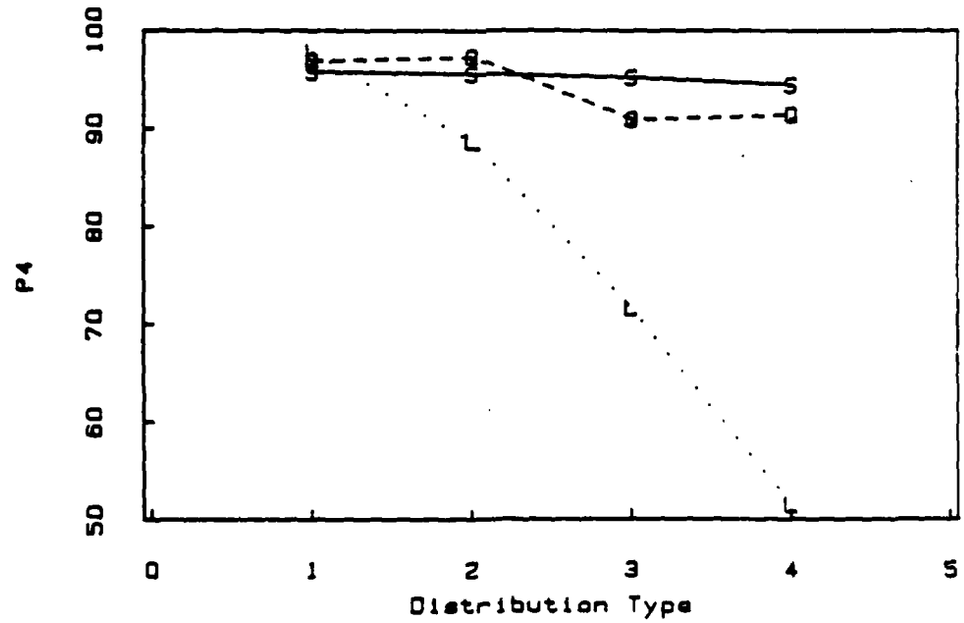


Figure 4.1.5. Percentage of test elements classified to same category with true and estimated P.P. (F.A.R.).
S--- Const. Spline, Q- - Quadratic, L... Linear.

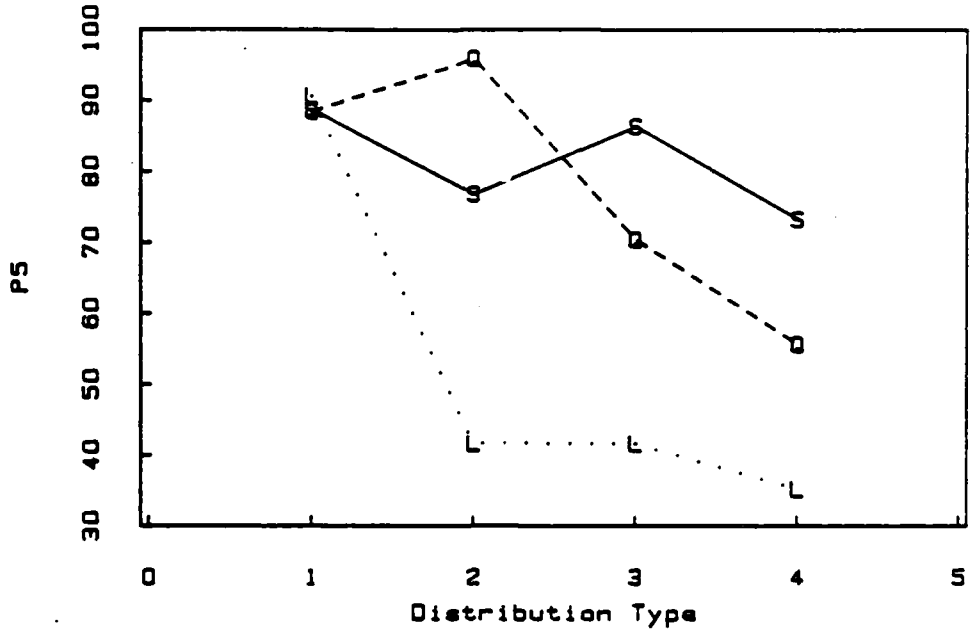


Figure 4.1.6. Percentage of test elements classified to same category with true and estimated P.P. (O.A.R.). S--- Const. Spline, Q- - Quadratic, L... Linear.

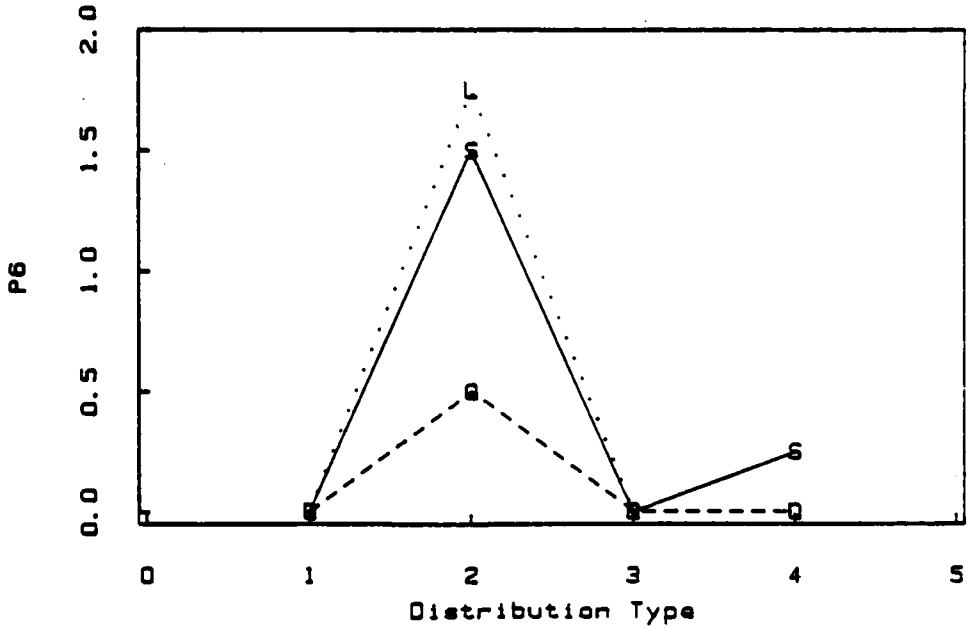


Figure 4.1.7. Percentage of strong disagreement between true and estimated Posterior Probability (O.A.R.) S--- Const. Spline, Q- - Quadratic, L... Linear.

performed better than the spline as expected, however the spline was very closed to the quadratic. In samples from distributions type 3 and 4 the spline was consistently better than the quadratic. The difference between the spline and the quadratic was larger when the doubt allocation rule was used.

The spline did particularly well in measures P4, P5 and P6 which give a better idea of how well the estimate approximates the true function.

From table 4.1.3 through 4.1.6 we can also see that in terms of classification there is not too much difference between the unconstrained and the constrained spline, being the performance of the constrained spline slightly better in most cases.

In the case of distribution type 3, six more samples were generated to see the effect of the sample size on the spline estimate. The first three simulations were done with $n_1 = n_2 = 25$ and the last three with $n_1 = n_2 = 45$. The results of these simulations together with the results for $n_1 = n_2 = 70$ are given in tables 4.1.7 and 4.1.8 and summarized in figures 4.1.8 through 4.1.13. In all the measures the spline performed better than the quadratic and linear models for the three different sample sizes.

In figures 4.1.14 to 4.1.33 we present, for one of the simulations for each type of distribution, plots of the approximate generalized cross-validation func-

TABLE 4.1.7

Distribution Type 3, Sample Size : 50

Perf. Measure	L	Stan. Dev.	Q	Stan. Dev.	S	Stan. Dev.	U.S.	Stan. Dev.
P1	67.67	3.18	82.83	1.26	85.50	2.18	86.33	0.76
P2	35.50	4.92	49.83	1.26	70.83	10.79	68.50	11.76
P3	1.67	2.47	1.67	1.61	3.50	2.78	3.17	2.25
P4	69.17	5.03	87.67	0.76	91.00	2.18	92.17	3.82
P5	44.50	1.73	69.50	3.50	76.17	3.75	79.67	2.93
P6	0.17	0.29	0.00	0.00	0.00	0.00	0.00	0.00

TABLE 4.1.8

Distribution Type 3, Sample Size : 90

Perf. Measure	L	Stan. Dev.	Q	Stan. Dev.	S	Stan. Dev.	U.S.	Stan. Dev.
P1	71.00	2.29	84.67	2.84	85.00	1.50	85.50	0.50
P2	36.83	2.36	52.33	2.36	59.17	5.69	56.33	6.25
P3	0.33	0.29	1.17	0.29	1.17	0.29	0.83	0.29
P4	71.67	1.26	93.00	0.50	93.33	2.89	94.17	2.75
P5	45.50	3.77	73.67	4.07	78.50	6.24	76.50	7.26
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

tion, contour plot of spline and quadratic estimates and the true posterior probability and surface plots for the true posterior probability and its spline estimate. In the case of distribution types 3 and 4 we also present contour plots of the quadratic estimate together with the true function.

In the plots of the generalized cross-validation function we use two different symbols: "+" and "o" to indicate when the set of active constraints changes. If for two consecutive values of λ the symbol changes, this indicates that the set of active constraints is different for those two values of λ.

In the contour plots we use a solid line for the contours of the constrained spline, dashed line (--) for the contours of the true posterior probability and a dash-dot line (-.-) for the contours of the quadratic estimate. The data is presented in the same plot. A dark triangle in the contour plot indicates the viewpoint for the surface plot.

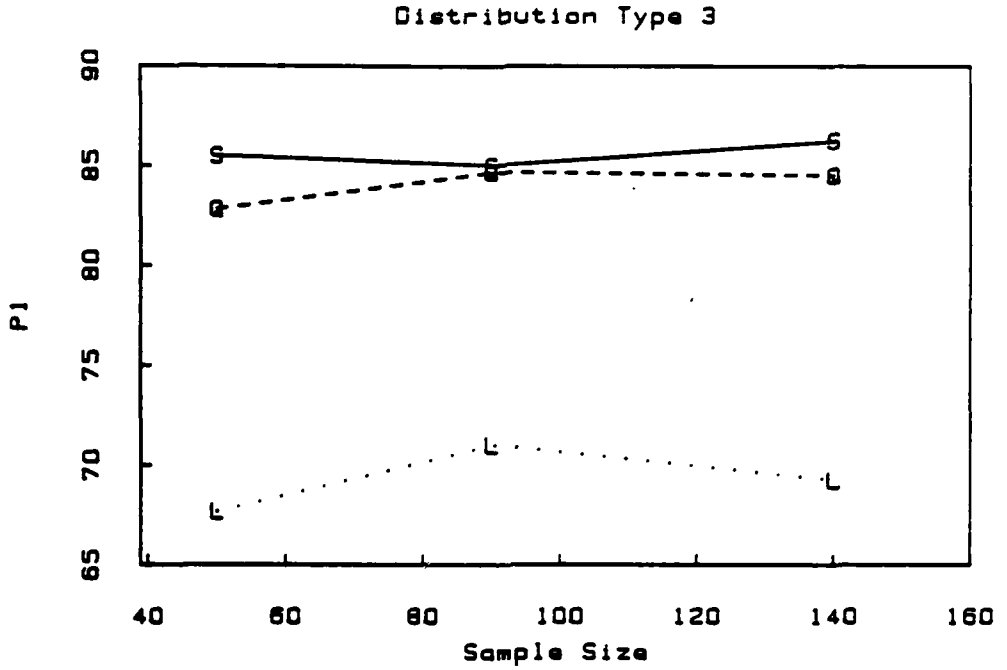


Figure 4.1.8: Percentage of test elements classified correctly using Forced Allocation Rule.
S--- Const. Spline, Q- - Quadratic, L... Linear.

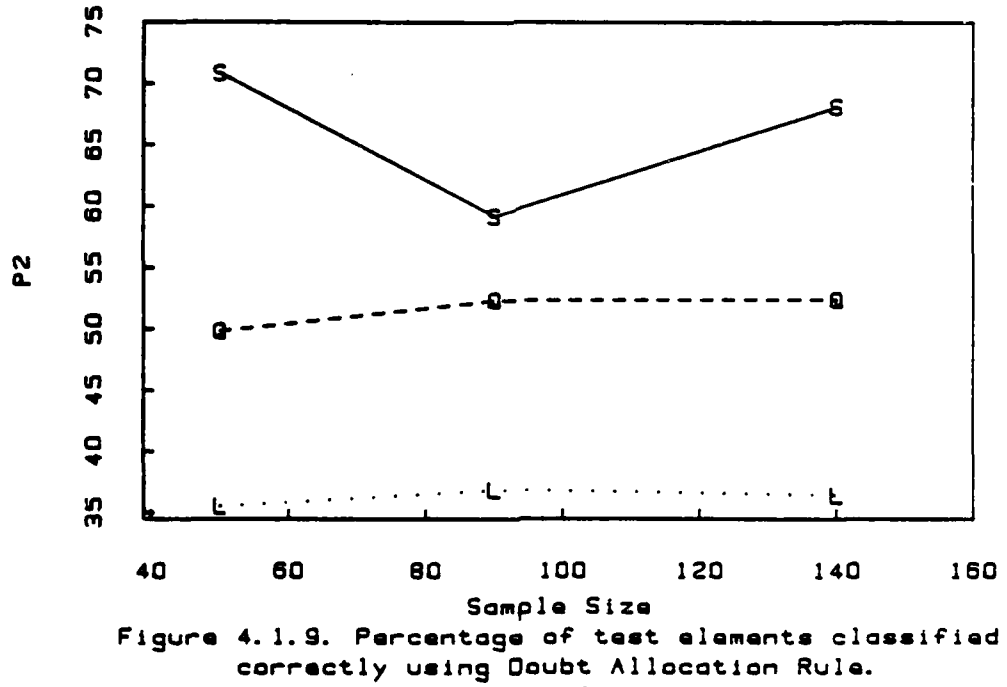


Figure 4.1.9. Percentage of test elements classified correctly using Doubt Allocation Rule.
S--- Const. Spline, Q- - Quadratic, L... Linear.

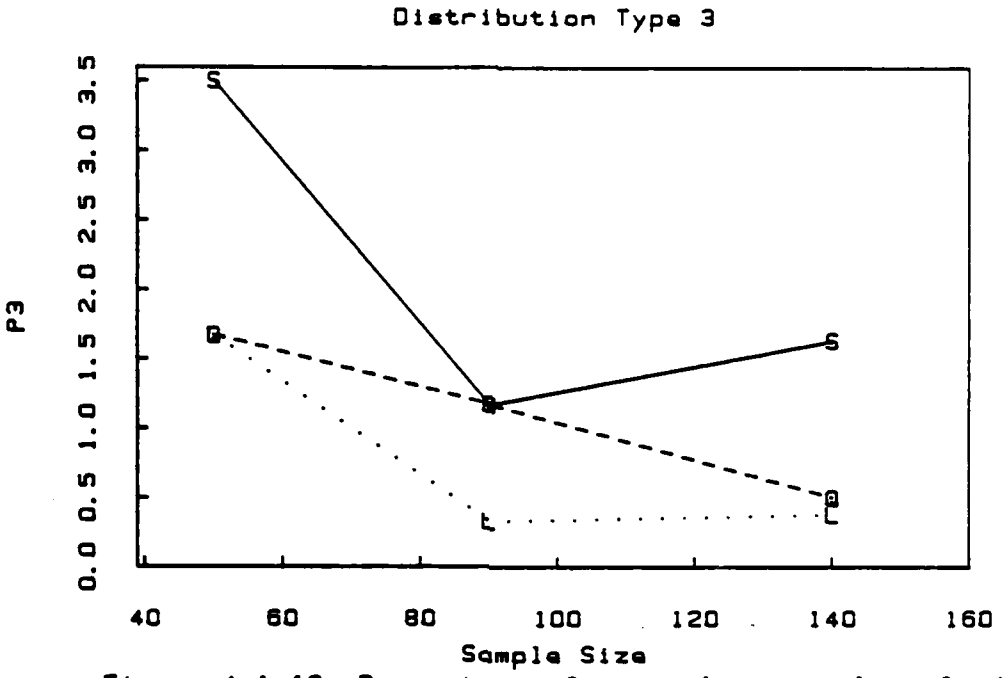


Figure 4.1.10. Percentage of test elements classified incorrectly using Doubt Allocation Rule.
S--- Const. Spline, Q- - Quadratic, L... Linear.

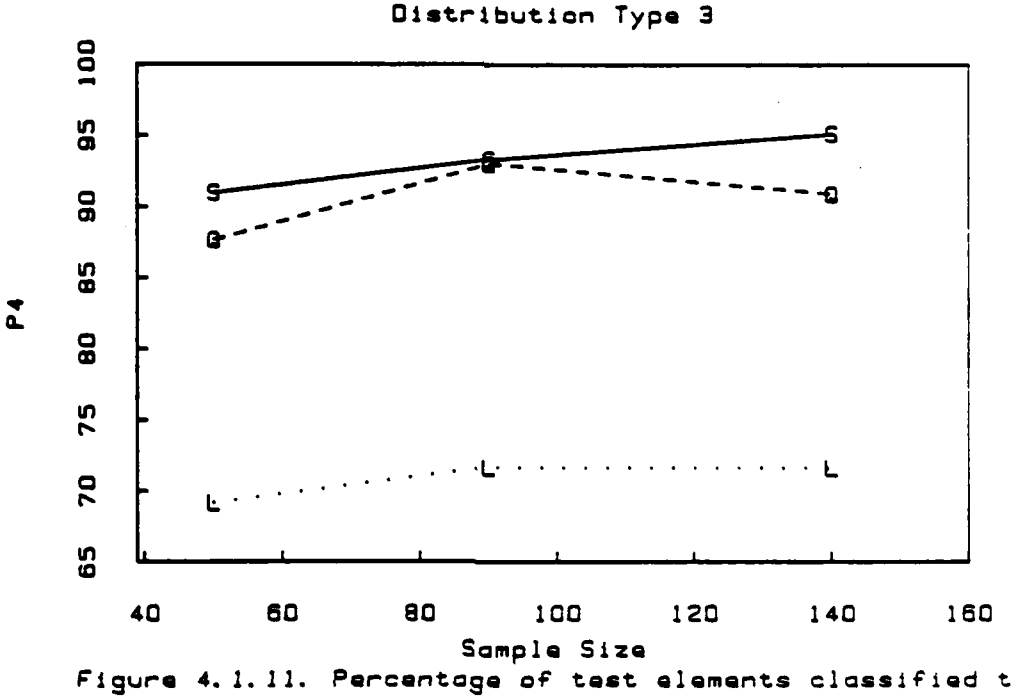


Figure 4.1.11. Percentage of test elements classified to same category with true and estimated P.P. (F.A.R.).
S--- Const. Spline, Q- - Quadratic, L... Linear.

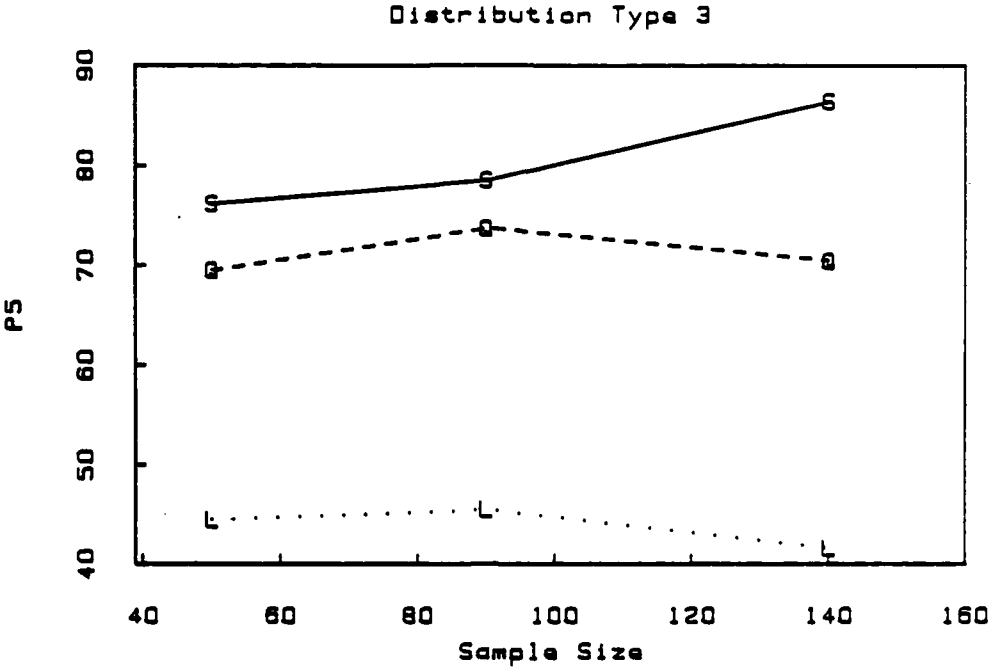


Figure 4.1.12. Percentage of test elements classified to same category with true and estimated P.P. (D.A.R.). S--- Const. Spline, Q- - Quadratic, L... Linear.

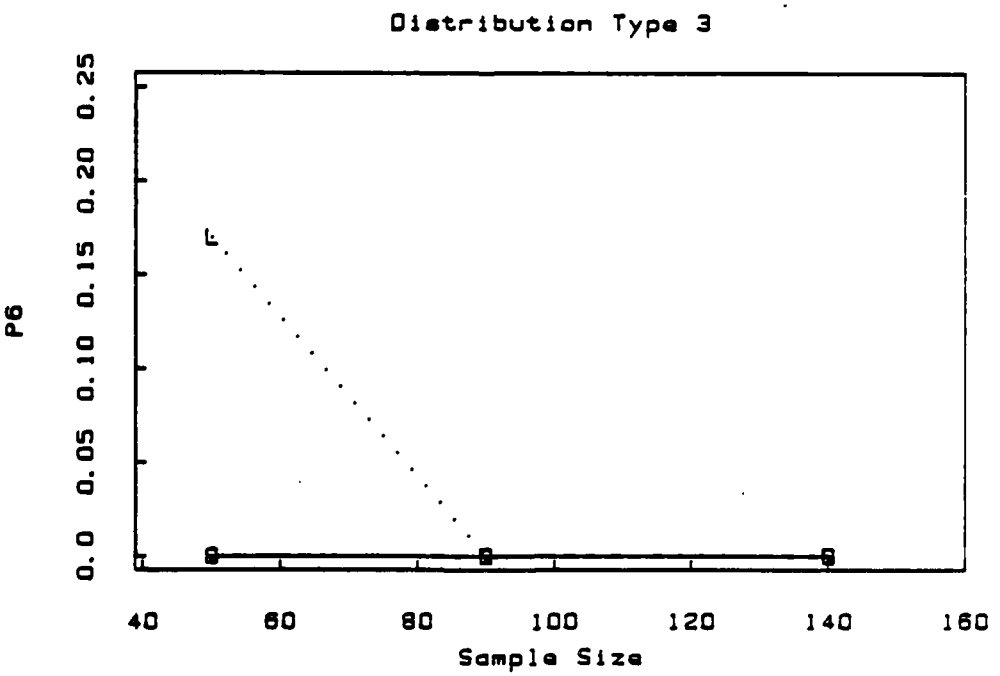


Figure 4.1.13. Percentage of strong disagreement between true and estimated Posterior Probability (D.A.R.) S--- Const. Spline, Q- - Quadratic, L... Linear.

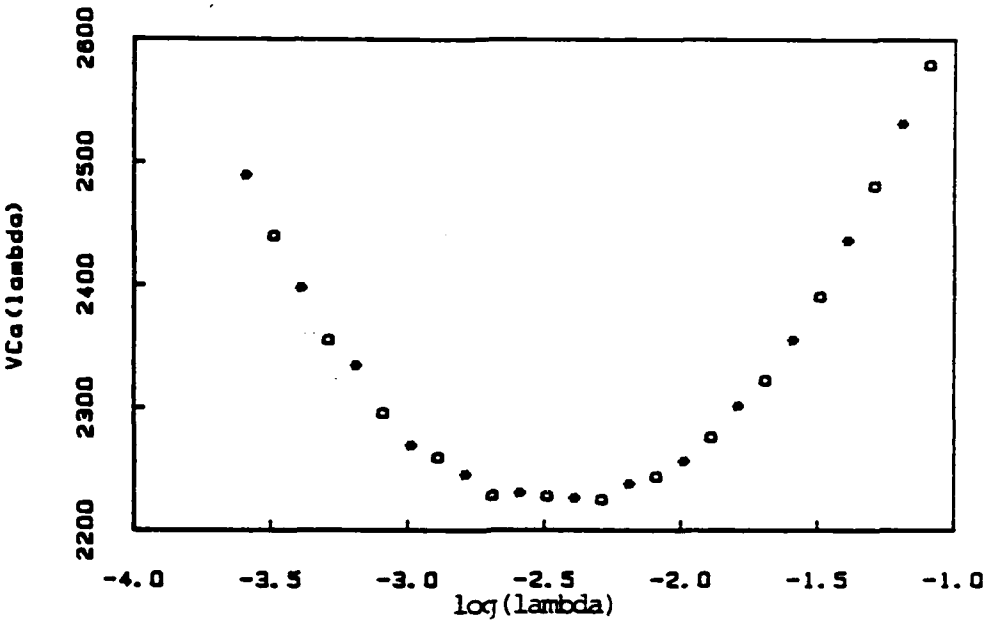


Figure 4.1.14: Approximate GCVC, Distribution type 1.

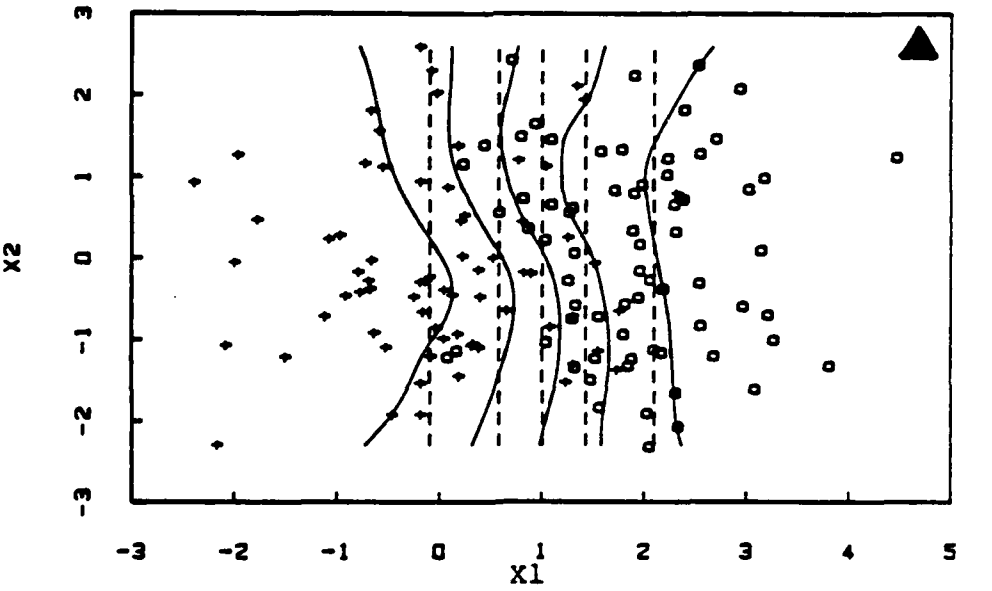


Figure 4.1.15: Constrained spline (——), true P.P. (- -) and data (+: sample 1, o: sample 2).

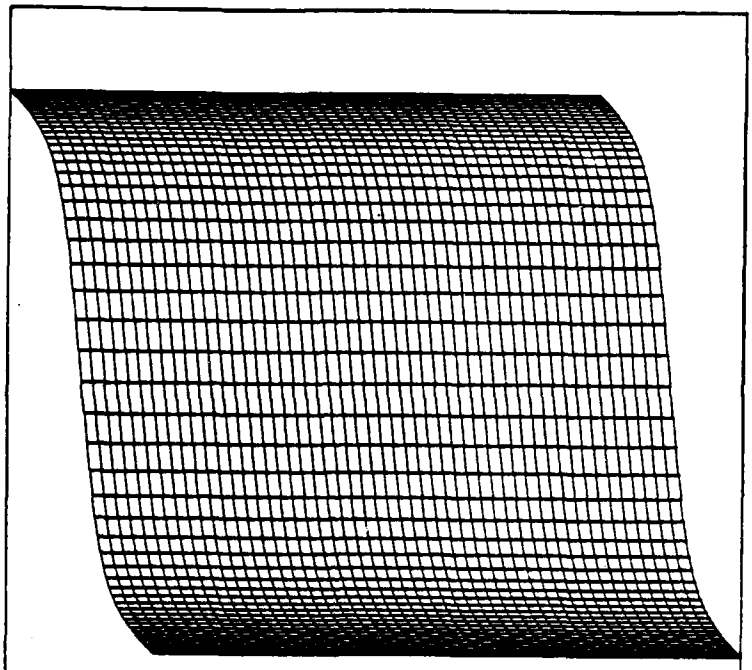


Figure 4.1.16: True Posterior Prob., distribution type 1.

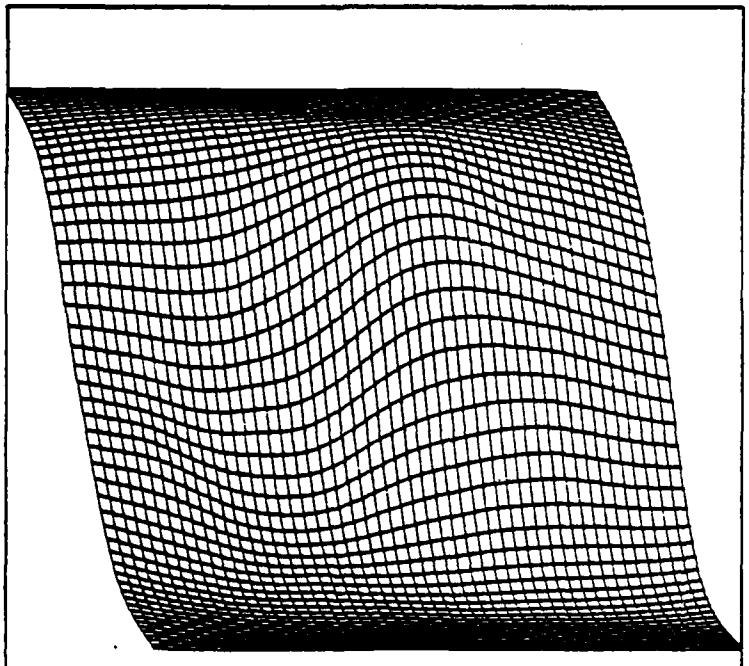


Figure 4.1.17: Constrained spline, distribution type 1.

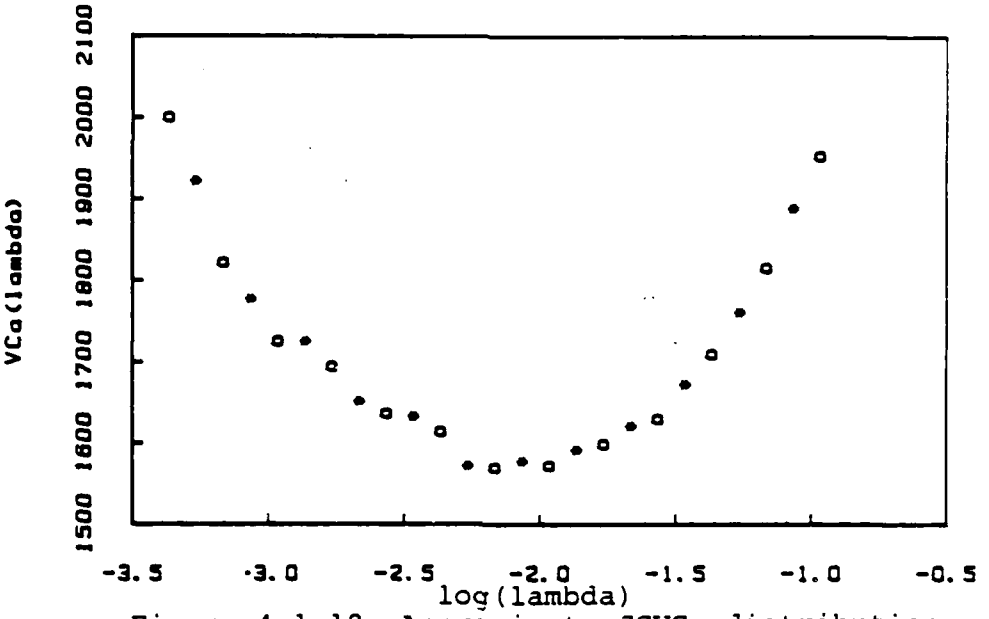


Figure 4.1.18: Approximate GCVC, distribution type 2.

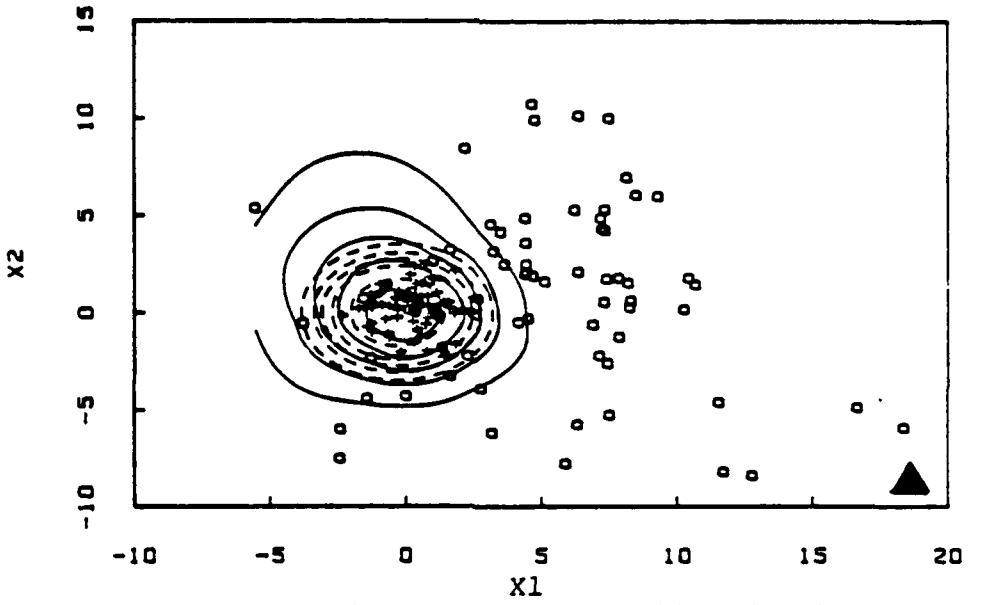


Figure 4.1.19: Constrained spline (——), true p. p. (- -) and data (+: sample 1, o: sample 2).

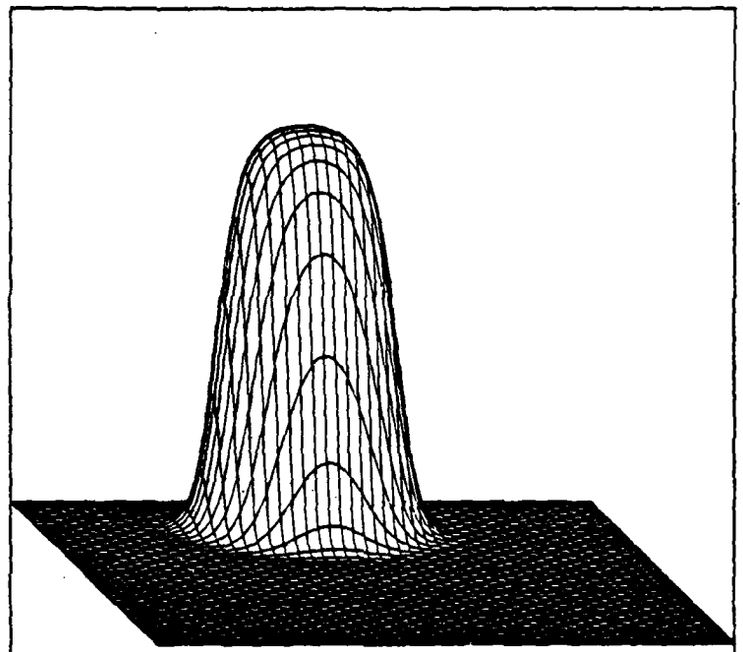


Figure 4.1.20: True posterior Prob. distribution type 2.

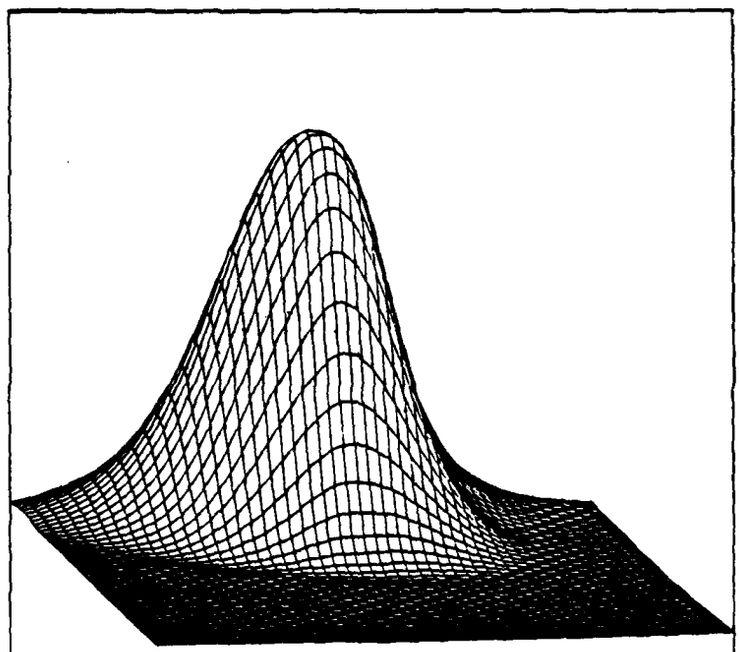


Figure 4.1.21: Constrained spline distribution type 2.

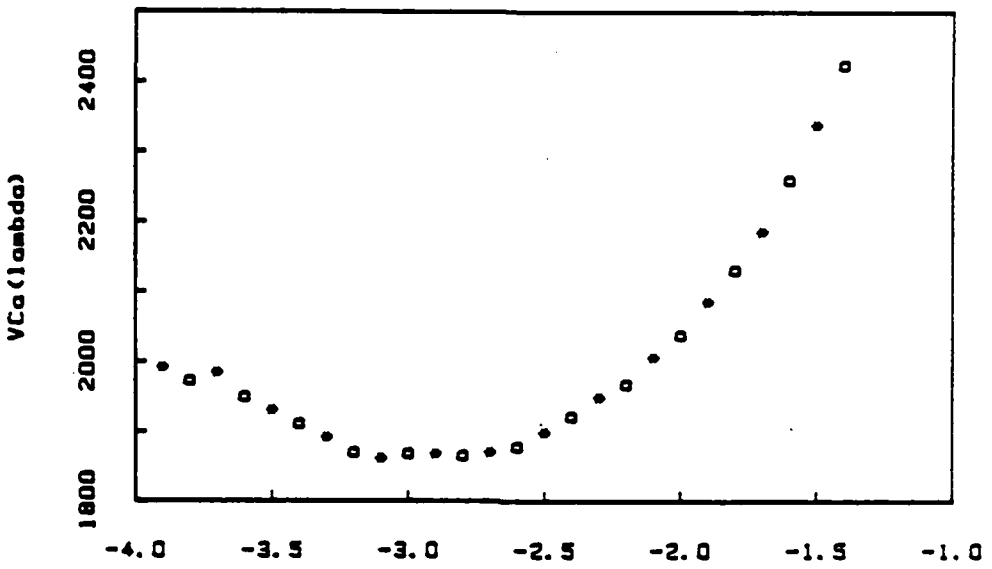


Figure 4.1.22: Approximate GCVC, distribution type 3.

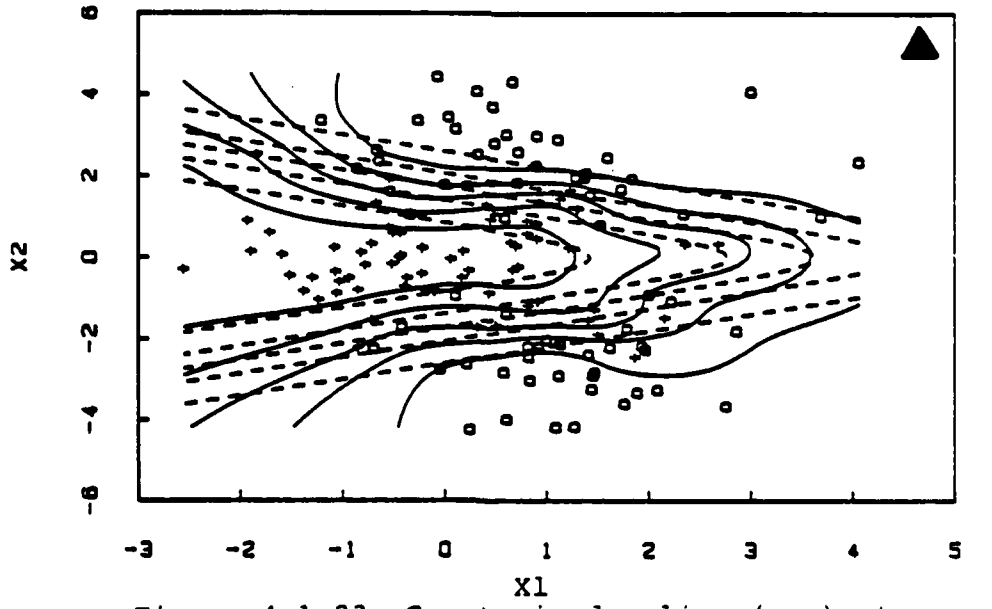


Figure 4.1.23: Constrained spline (——), true P. P.(- -) and data (+: sample 1, o: sample 2).

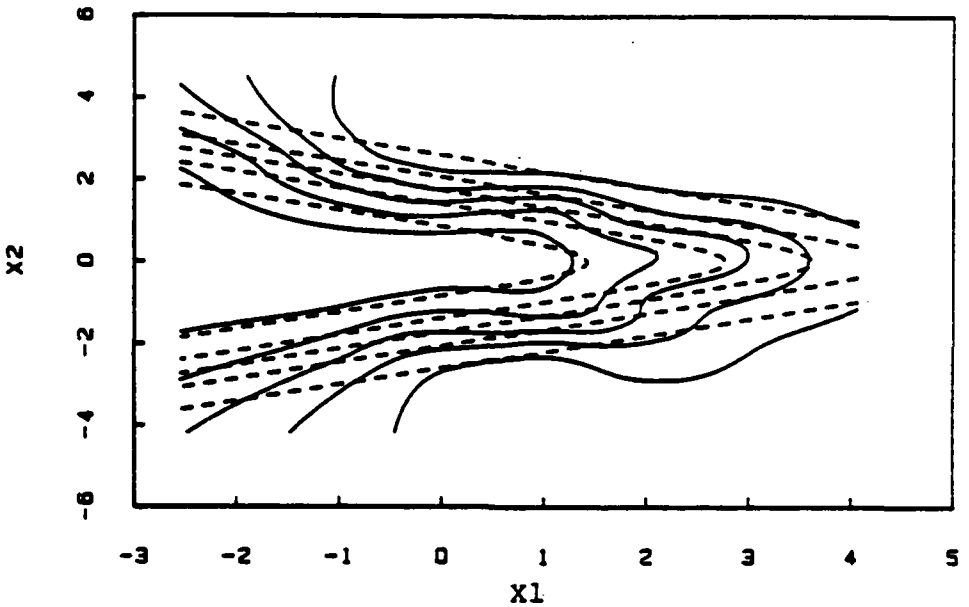


Figure 4.1.24: Constrained spline (——) and true (- -).

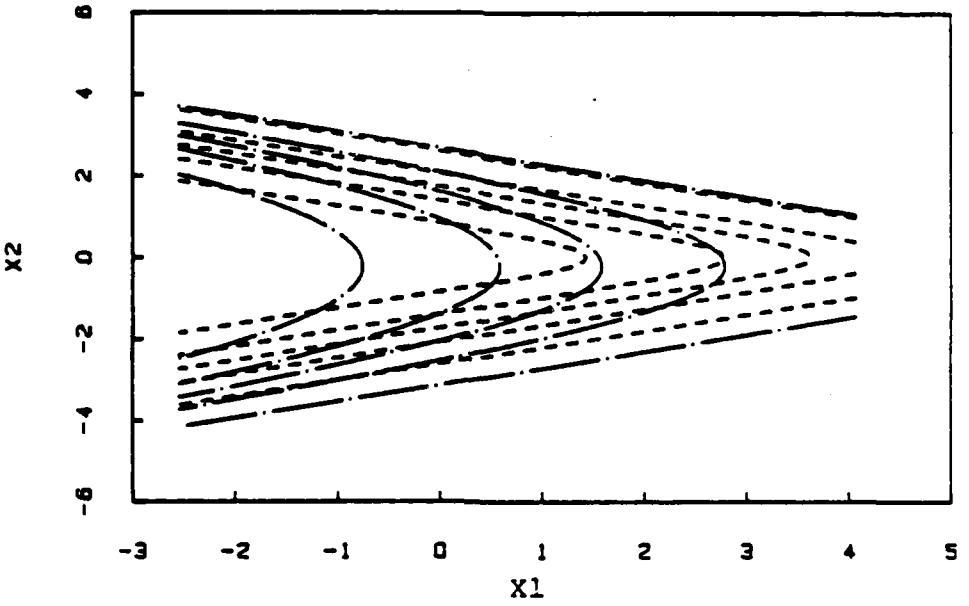


Figure 4.1.25: Quadratic estimate (-·-) and true (- -).

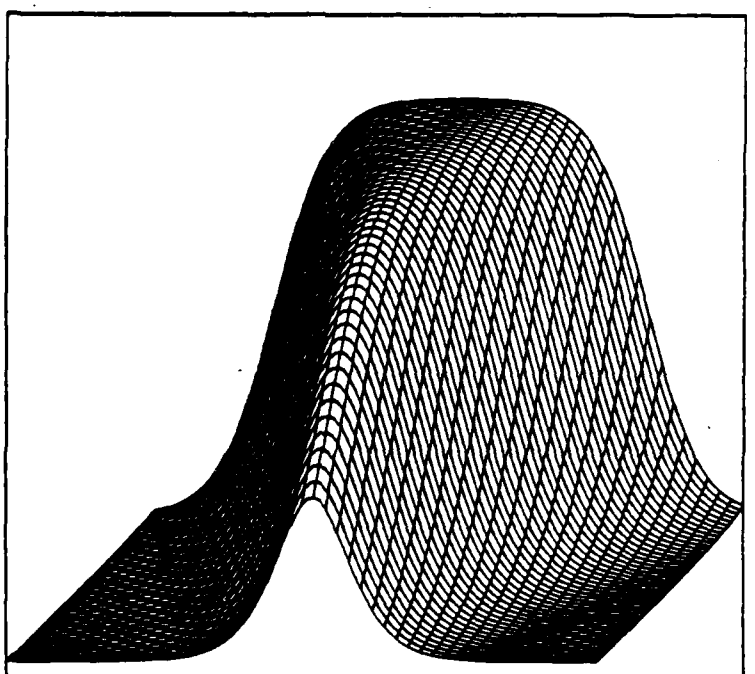


Figure 4.1.26: True posterior Prob. distribution type 3.

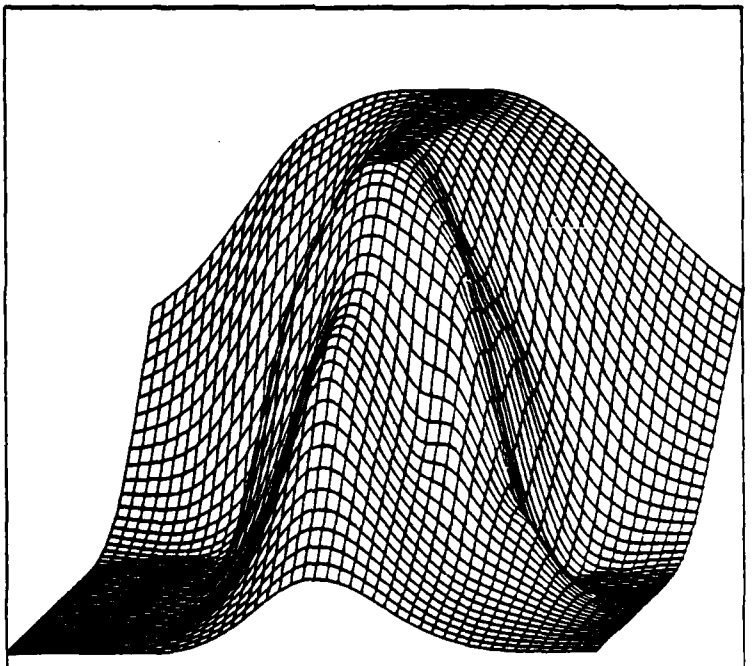


Figure 4.1.27: Constrained spline , distribution type 3.

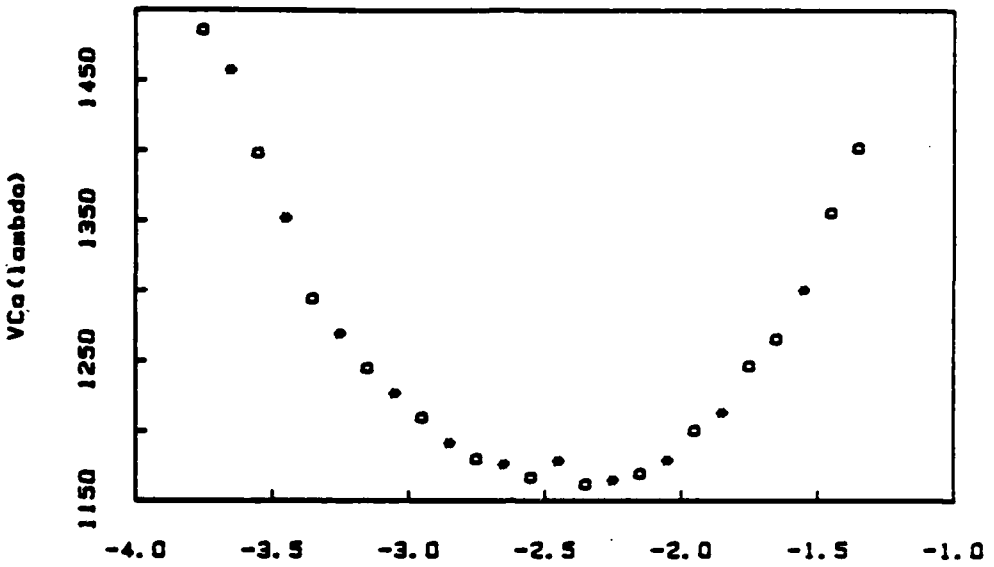


Figure 4.1.28: Approximate GCVC, distribution type 4.

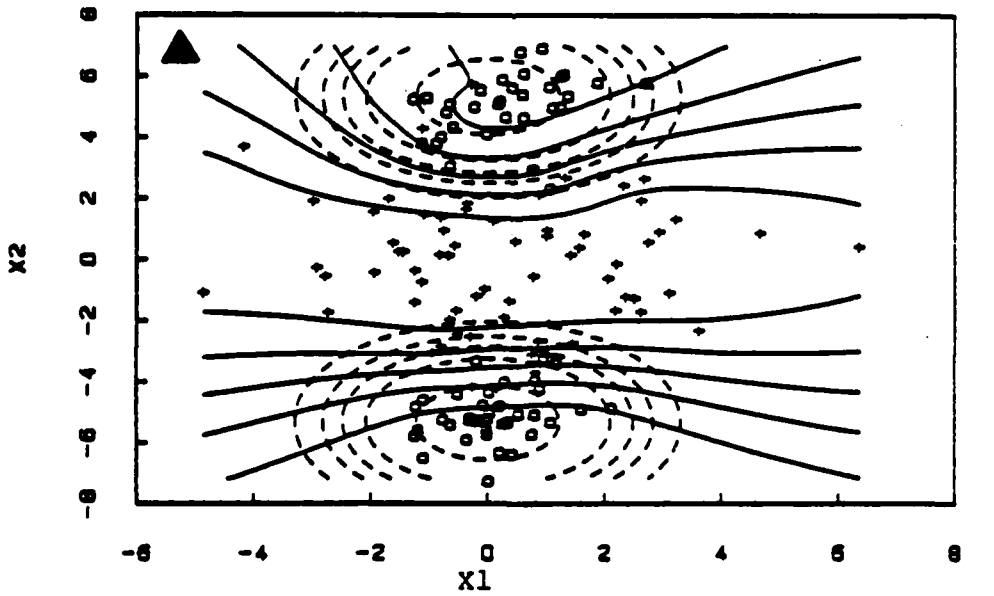


Figure 4.1.29: Constrained spline (——), true P.P.(- -) and data (+: sample 1, o: sample 2).

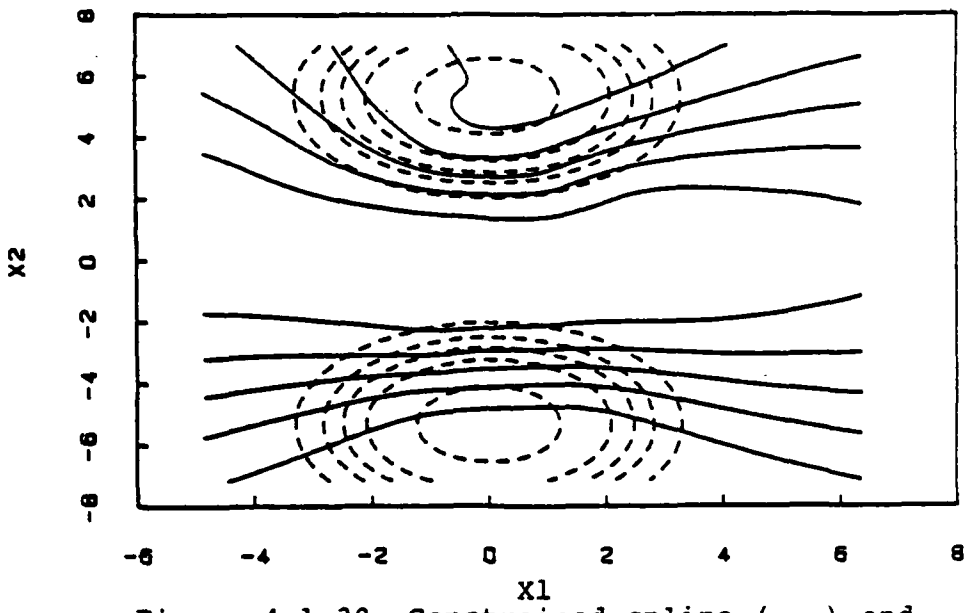


Figure 4.1.30: Constrained spline (——) and true (- -).

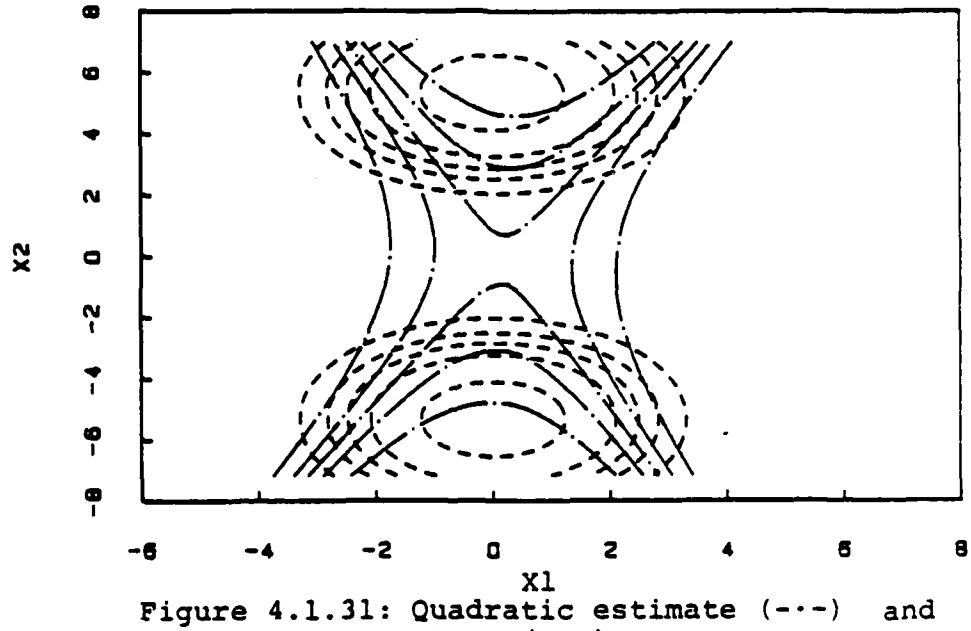


Figure 4.1.31: Quadratic estimate (-·-) and true (- -).

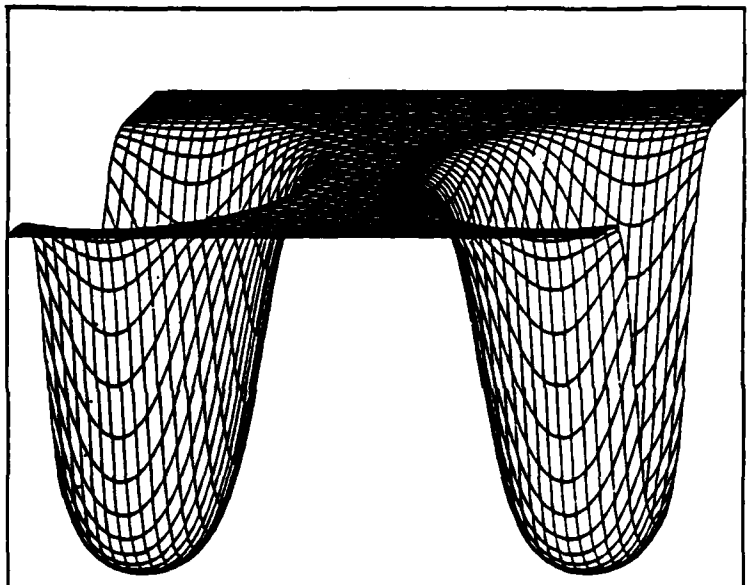


Figure 4.1.32: True posterior Prob., distribution type 4.

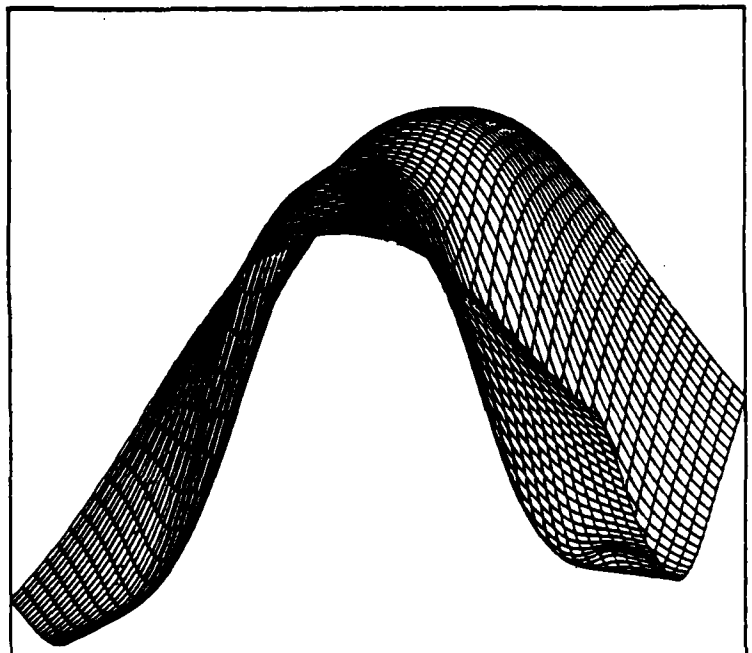


Figure 4.1.33: Constrained spline, distribution type 4.

4.2 Other simulation results

In this section we present two examples that were generated from univariate distributions and we give some computation times for the simulations of section 4.1.

In our first univariate example, sample 1 was generated from a Normal distribution with mean zero and variance one, and sample 2 from a Normal distribution with mean 1.5 and variance 1. In figures 4.2.1 and 4.2.2 we present the unconstrained and constrained spline estimates, together with the quadratic estimate, true posterior probability and data.

In the second univariate example, sample 1 was generated from $0.3N(0,0.25)+0.7N(0,16)$, and sample 2 from a $N(0,0.25)$. The corresponding plots are given in figures 4.2.3 and 4.2.4.

In figures 4.2.2, 4.2.3, 4.2.5, and 4.2.6 it is clear that for classification purposes there is basically no difference between the unconstrained and the constrained splines, but certainly, we would not want to show a client a plot of an estimated posterior probability that can take values greater than one or smaller than zero.

The time of computation of the spline depends on the sample size $n=n_1+n_2$, the number of constraints k, the number of values of λ to be considered for evaluation of the generalized cross-validation function and the number of times that the active set of constraints changes from one value of λ to the next.

In table 4.2.1 we present the average C.P.U time used by the routine DSCOMP to compute the spline in the simulations of section 4.1. In table 4.2.1 the column labeled "nc" contains the average number of constraints enforced in the simulations, the column labeled "%Time in G.E.P." contains the percentage

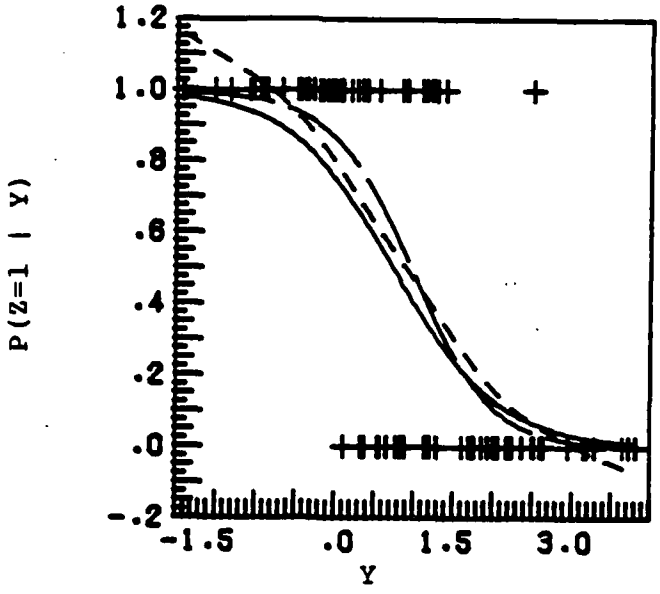


Figure 4.2.1: Unconstrained spline (- - -), quadratic (— —), true (——) and data. $n_1=n_2=40$

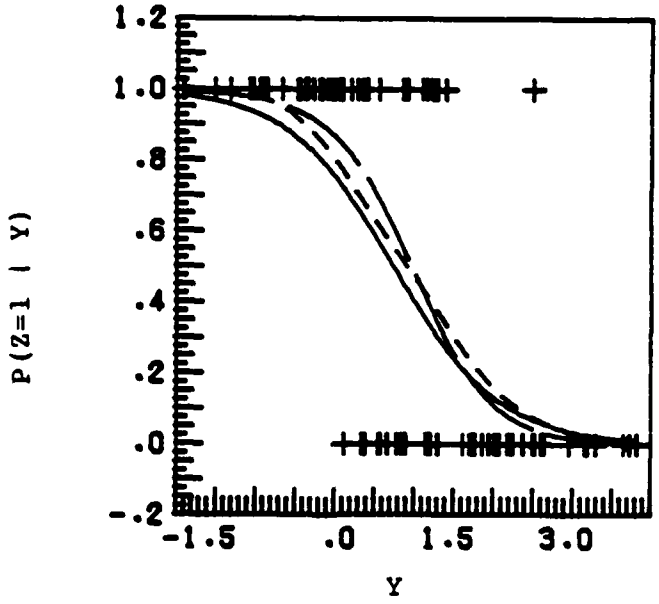


Figure 4.2.2: Constrained spline (- - -), quadratic (— —), true (——) and data. $n_1=n_2=40$

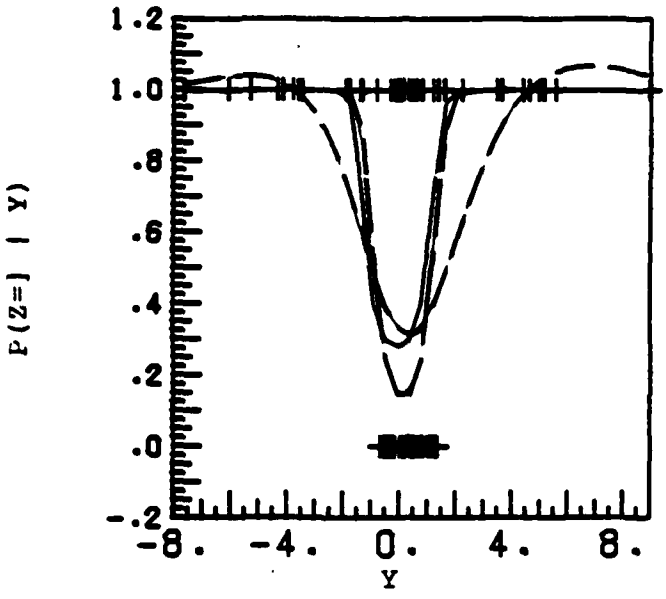


Figure 4.2.3: Unconstrained spline (- - -), quadratic (— —), true (——) and data. $n_1=n_2=45$.

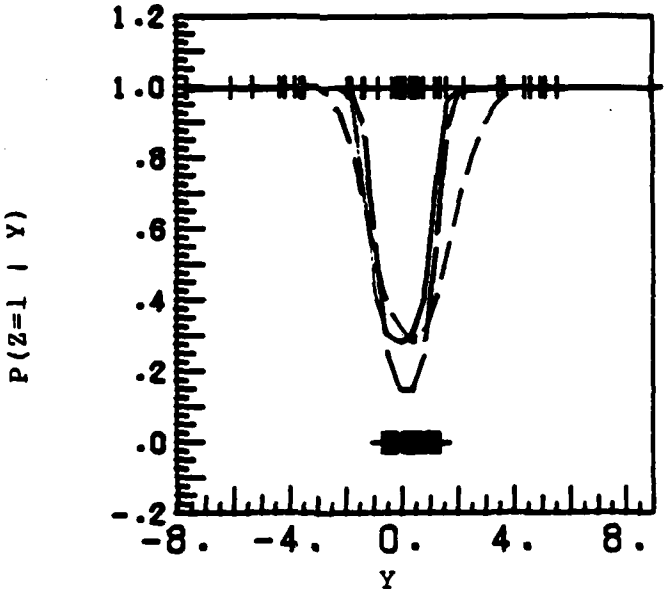


Figure 4.2.4: Constrained spline (- - -), quadratic (— —), true (——) and data. $n_1=n_2=45$.

of time spent in the generalized eigenvalue problem and the column labeled "%Time in Q.P." contains the percentage of time spent in the solution of the quadratic programming problem.

TABLE 4.2.1

Dist. Type	n	nc	CPU time (minutes)	%Time in G. E. P.	%Time in Q. P.
1	140	98	252.03	43.5	40.2
2	140	94	230.45	43.1	46.0
3	50	78	54.15	21.6	62.9
3	90	65	63.33	32.3	56.7
3	140	67	184.24	54.1	34.9
4	140	95	293.11	47.5	41.7

From this table we can see that on the average 47% of the computation time is spent on the solution to the generalized eigenvalue problem and 40% in the solution of the quadratic programming problem except in the small sample cases (n=50, 90). In the n=50 case 62.9% of the CPU time was spent in the solution of the quadratic programming problem while only 21.6% of the time was spent in the generalized eigenvalue problem. In the n=90 case, 56.7% of the time was spent in the quadratic programming problem and 32.3% in the generalized eigenvalue problem. The reason for this is that the quadratic programming routine works better when the number of constraints in the active set is small compared with the number of observations. The algorithm to solve the generalized eigenvalue problem is an N^3 algorithm, where $n-M \le N \le n+k-M$, therefore it would be very expensive to use the routine DSCOMP to compute the constrained spline for very large n. In our simulation study even with n=50 we got pretty good results and in distributions type 3 and 4 the spline was consistently better than the quadratic model for n=50, 90 and 140, however we do not recommend its use for very small n. In chapter 6 we will discuss a possible alternative for large sample sizes.

4.3 An example

Here we apply our method of estimating the posterior probabilities to some results of a psychological test of a group of 25 normal persons and 25 psychotics. We obtained this set of data from Smith (1947). The psychotics will be population A_1 and the normals A_2. The two covariates are an unweighted total score or "size" obtained by Penrose's method (Penrose 1945) and a weighted score related to shape.

In figure 4.3.2 we present the data and the contour levels for the spline estimate (solid lines) of the probability that a given subject belongs to the psychotic group given its particular measurements of size and shape. From this figure it appears that the spline estimate gives a more accurate representation of the data than the quadratic estimate.

In figure 4.3.3 we present a view of the surface of the spline estimate from the point indicated by a dark triangle in the contour plot 4.3.2. The approximate generalized cross-validation function is given in figure 4.3.1.

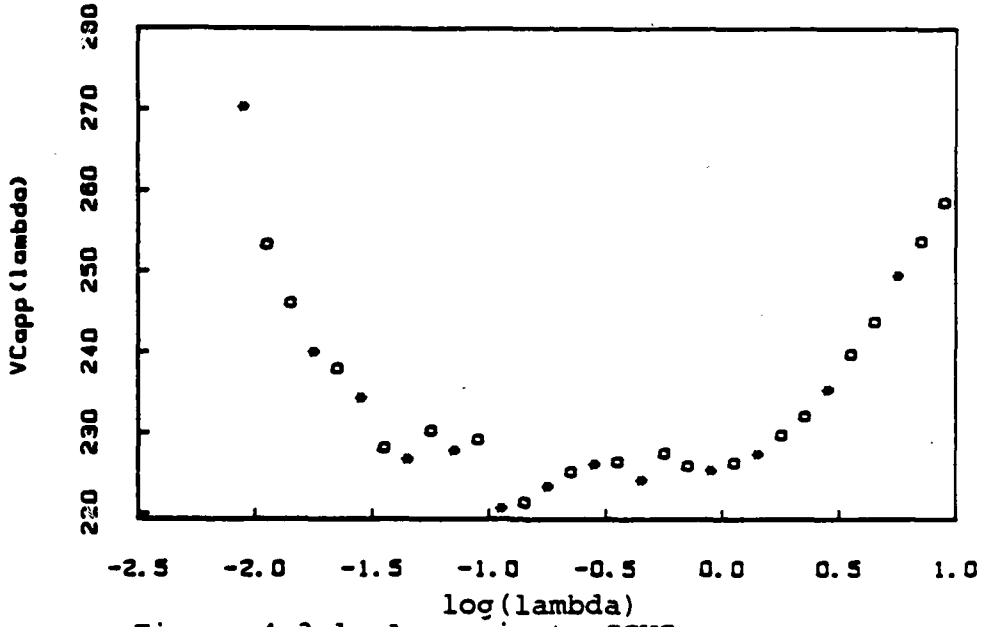


Figure 4.3.1: Approximate GCVC.

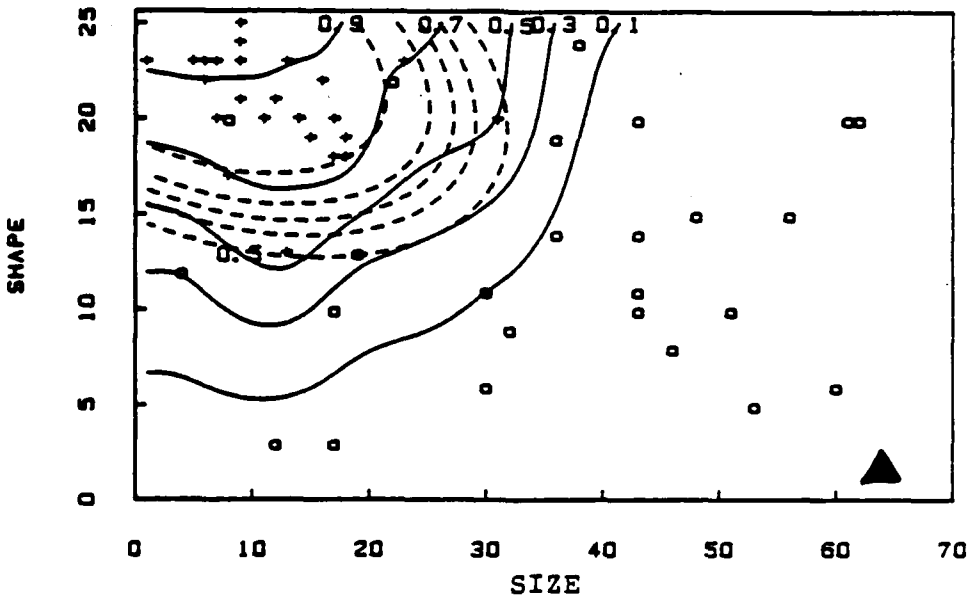


Figure 4.3.2: Constrained spline (——) and quadratic (- -). +: Psychotics, o: Normals.

constrained spline

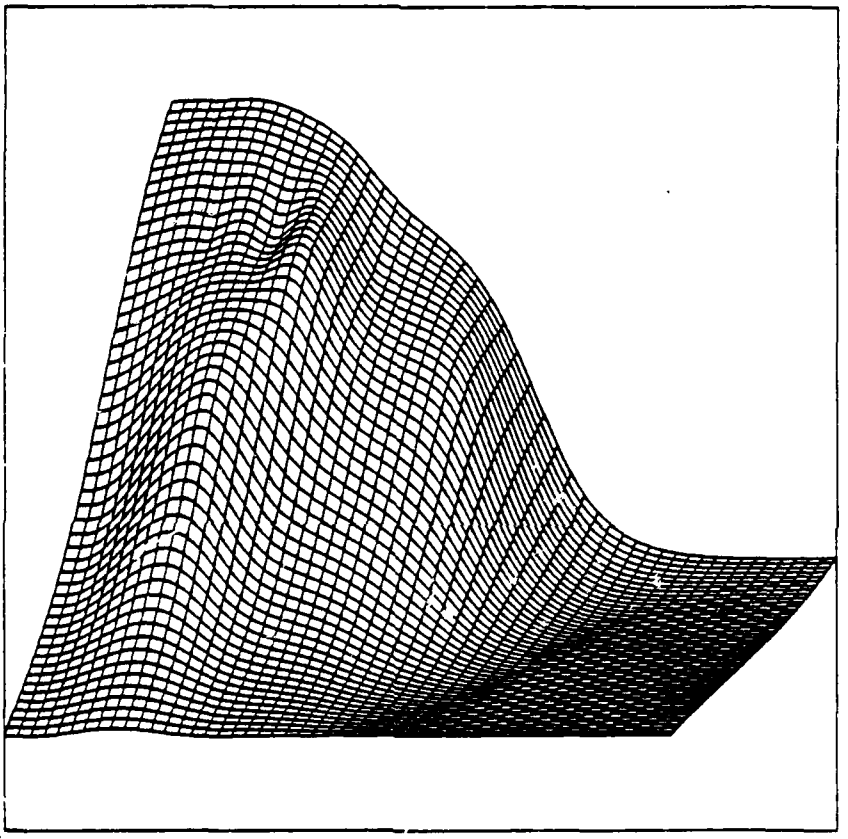


Figure 4.3.3

CHAPTER 5

PENALIZED LIKELIHOOD ESTIMATION

5.1 Motivation

In this chapter we extend the result given by Silverman (1978) to the d-dimensional case.

As in chapter 2 let $Y_1, \ldots, Y_n$, $n=n_1+n_2$ denote the combined samples from the two populations and define

$$Z_i = \begin{cases} 1 \text{ if } Y_i \in A_1 \\ 0 \text{ if } Y_i \in A_2 \end{cases} \tag{5.1.1}$$

Let q_1 and q_2 be the prior probabilities. Then, if we obtain a new observation Y and we want to classify it as comming from A_1 or A_2, the Neyman-Pearson lemma tells us that the classification rule should be based in the ratio

$$r = \frac{q_1 f_1}{q_2 f_2} \tag{5.1.2}$$

Besides its application in the classification problem, the estimation of density ratios lays open the possibility of constructing empirical versions of any other procedure based on likelihood ratios.

We now derive the likelihood of r conditional on the observed values $z_1, \ldots, z_n$ and $y_1, \ldots, y_n$.

Note that

$$P(Z_i \mid Y_i) = \frac{q_1 f_1(y_i)}{q_1 f_1(y_i)+q_2 f_2(y_i)} = \frac{r(y_i)}{1+r(y_i)}, \tag{5.1.3}$$

and similarly:

$$P(Z_i=0 \mid Y_i=y_i) = \frac{1}{1+r(y_i)}. \tag{5.1.4}$$

So that if $z_1, \ldots, z_n$ are the observed values of $Z_1, \ldots, Z_n$, the conditional likelihood for r is given by

$$L(r) = \prod_{i=1}^{n} \frac{\left[r(y_i)\right]^{z_i}}{1+r(y_i)}, \tag{5.1.5}$$

and hence, the log-likelihood is given by

$$\log L(r) = \sum_{i=1}^{n} \left\{ z_i \log r(y_i) - \log[1+r(y_i)] \right\}. \tag{5.1.6}$$

A maximum likelihood approach to estimate r would be to maximize (5.1.6). The fact that $L(r)$ is undefined for negative values of $r(y_i)$, while, for any i such that z_i is zero, the likelihood increases as $r(y_i)$ tends to zero can lead to computational difficulties if the estimation of r is considered directly. To avoid this, consider the estimation of the logarithm of r and let

$$g = \log r. \tag{5.1.7}$$

then the conditional likelihood L^* of g becomes

$$L^*(g) = \prod_{i=1}^{n} \frac{\left\{\exp[g(y_i)]\right\}^{z_i}}{1+\exp[g(y_i)]}$$

and the log-likelihood is

$$\log L^*(g) = \sum_{i=1}^{n} \left[z_i g(y_i) - \log\left\{1+exp\left[g(y_i)\right]\right\} \right]. \tag{5.1.8}$$

If we wanted to maximize (5.1.8), we could take

$$g(y_i) = \begin{cases} \infty & \text{if } z_i = 1 \\ -\infty & \text{if } z_i = 0 \end{cases}.$$

To avoid this undesirable solution we should use the underlying assumption that g is, in some sense, nc too rough. Therefore we should penalize the likelihood according to the roughness of g.

This device of penalizing for roughness has been used in different contexts by Reinsch (1967), Good and Gaskins (1971) and other subsequent authors.

We will assume that the function g is in the reproducing kernel Hilbert space $H(m,d)$ defined in section 2.2. The "penalized likelihood" estimate of g is the function that minimizes

$$I_\lambda(g) = -\log L^*(g)+\lambda J_m(g), \tag{5.1.9}$$

where J_m is given by (2.2.8). In the particular case where $m=2$ and $d=1$, $I_\lambda(g)$ becomes

$$\sum_{i=1}^{n}\left[\log(1+exp[g(y_i)])-z_i g(y_i)\right] + \lambda\int(g'')^2,$$

which is the expression that Silverman (1978) minimizes to estimate g.

5.2 Existence and uniqueness of an estimator

In this section we discuss the existence and uniqueness of the minimizer of $I_\lambda(g)$ in $H(m,d)$, and we characterize it as a Laplacian Smoothing Spline. The optimization theoretic results needed here are given in appendix A1.

As in chapter 2, let $H_0(m,d)$ be the space of polynomials of degree less than m defined on $\mathbf{R}^d$, and let $H_1(m,d)$ be the orthogonal complement of $H_0(m,d)$ in $H_1(m,d)$.

First we establish the uniqueness of a minimizer of $I_\lambda(g)$ in the following

5.2.1 Lemma

The minimizer $\hat{g}_\lambda$ in $H(m,d)$ of $I_\lambda(g)$, if it exists is unique.

Proof

Let $f \in H(m,d)$ and define

$$\zeta(t) := I_\lambda(g+tf)$$

then, by proposition A1.4, we have that the j^{th} Gateaux variation of $I_\lambda(g)$ in the direction $(f,\ldots,f)$ is given by

$$I_\lambda^{(j)}(g)(f,\ldots,f) = \frac{d^j}{dt^j}\zeta(t)\Big|_{t=0}.$$

Note that $J_m(g) = \|Pg\|_H^2$ where P is a projection operator into the space $H_1(m,d)$, then, we compute the first and second Gateaux variations as follows:

$$\begin{aligned}\frac{d}{dt}\zeta(t) &= \frac{d}{dt}\left\{\sum_{i=1}^{n} z_i[g(y_i)+tf(y_i)] - \log(1+\exp[g(y_i)+tf(y_i)])\right\} \\ &\quad + \frac{d}{dt}\left\{\lambda\langle g+tf, g+tf\rangle\right\} \\ &= \sum_{i=1}^{n} z_i f(y_i) - \frac{f(y_i)\exp[g(y_i)+tf(y_i)]}{1+exp[g(y_i)+tf(y_i)]} \\ &\quad + 2\lambda\langle f, g\rangle + 2\lambda t\langle f, f\rangle\end{aligned}$$

And differentiating with respect to t again we obtain:

$$\frac{d^2}{dt^2}\zeta(t) = \sum_{i=1}^{n} \frac{f(y_i)\exp[g(y_i)+tf(y_i)]}{1+exp[g(y_i)+tf(y_i)]} + 2\lambda\langle f, f\rangle.$$

Finally, evaluating at t=0 we get:

$$\begin{aligned}I_\lambda^{(2)}(g)(f,f) &= \sum_{i=1}^{n} \frac{f(y_i)^2 exp[g(y_i)]}{1+\exp[g(y_i)]} + 2\lambda\|Pf\|^2 \\ &> 0 \;\forall\; f \in H(m,d), f \neq 0.\end{aligned}$$

Then, by proposition A1.5 I_λ is strictly convex, and hence, if it has a minimum, it is unique.

■

We now establish sufficient and necessary conditions for the existence of a minimizer of I_λ in the space $H_0(m,d)$.

5.2.2 Lemma

Let $H_0(m,d)$ be the space of polynomials of degree less than m defined in $\mathbf{R}^d$. The minimizer of I_λ exists and is unique if and only if there is no level curve of an element of $H_0(m,d)$ that completely separates the samples.

Proof

Let $g \in H_0(m,d)$, then, since $J_m(g) = 0$,

$$I_\lambda(g) = \sum_{i=1}^{n} \left\{ \log(1+exp[g(y_i)] - z_i g(y_i) \right\}.$$

Now, there exists a level curve of a polynomial that completely separates the samples if and only if there exists $g^* \in H_0(m,d)$ such that

$$g^*(y_i) = \begin{cases} >0 & \text{if } y_i \in A_1 \\ <0 & \text{if } y_i \in A_2 \end{cases} \tag{5.2.1}$$

We first show the if part. Suppose that there is no $g \in H_0(m,d)$ satisfying (5.2.1), then for all $g \in H_0(m,d)$ there exists at least one y_j, $j=1, \ldots, n$ such that

$$g(y_j) \approx \begin{cases} <0 & \text{if } y_j \in A_1 \\ >0 & \text{if } y_j \in A_2 \end{cases}$$

Now,

$$I_\lambda(g) = \sum_{i=1}^{n} \left\{ log(1+exp[g(y_i)] - z_i g(y_i) \right\}$$

$$\geq \log(1+exp[g(y_i)] - z_j g(y_j)).$$

So, if $y_j \in A_1$, $g(y_j)<0 \Rightarrow I_\lambda(g) \to \infty$ as $\|g\|_{H_0} \to \infty$ and if $y_j \in A_2$, $g(y_j)>0 \Rightarrow I_\lambda(g) \to \infty$ as $\|g\|_{H_0} \to \infty$. Therefore, by proposition A1.7 there exists $\hat{g} \in H_0(m,d)$ that minimizes $I_\lambda(g)$ in $H_0(m,d)$ and by Lemma 5.2.1 such $\hat{g}$ is unique.

We now show the only if part: assume that there is a unique $\hat{g} \in H_0(m,d)$ that minimizes I_λ and suppose that there exists g^* that completely separates the samples, that is, g^* satisfies (5.2.1). Since $\hat{g}$ minimizes I_λ, $\hat{g}$ maximizes $\exp[-I_\lambda(g)]$

$$\Rightarrow \exp[-I_\lambda(\hat{g})] \geq exp[-I_\lambda(g)] \; \forall \; g \in H_0(m,d).$$

But

$$exp[-I_\lambda(g)] = \prod_{z_i=1} \frac{\exp[g(y_i)]}{1+\exp[g(y_i)]} \prod_{z_i=0} \frac{\exp[-g(y_i)]}{1+\exp[-g(y_i)]}$$

and looking at each term in the above expression we see that $exp[r]/(1+exp[r])$ goes from 0 to $\frac{1}{2}$ when r goes from $-\infty$ to 0 and it goes to 1 when r goes from 0 to ∞ and necessarily $exp[-I_\lambda(g)] < 1$. Now, for arbitrary $\vartheta > 0$ consider

$$\exp[-I_\lambda(\vartheta g^*)] = \prod_{z_i=1} \frac{exp[\vartheta g^*(y_i)]}{1+exp[\vartheta g^*(y_i)]} \prod_{z_i=0} \frac{\exp[-\vartheta g^*(y_i)]}{1+exp[\vartheta g^*(y_i)]}$$

Since g^* satisfies (5.2.1) both products tend to 1 as $\vartheta \to \infty$, so that for ϑ sufficiently large we have

$$exp[-I_\lambda(\vartheta g^*)] > exp[-I_\lambda(\hat{g})],$$

which contradicts the assumption that $\hat{g}$ is the unique minimizer of $I_\lambda(g)$.

■

In Theorems 5.2.1 and 5.2.2 below, we establish conditions for existence and uniqueness of the minimizer of $I_\lambda(g)$ in $H(m,d)$ and characterize the solution as a Laplacian smoothing spline.

5.2.1 Theorem

The minimizer $\hat{g}_\lambda$ of $I_\lambda(g)$ in $H(m,d)$ exists and is unique provided that there is no level curve of a polynomial of degree less than m that completely separates the samples.

Proof

Uniqueness follows from lemma 5.2.1. By lemma 5.2.2 we only need to show that if the minimizer of $I_\lambda(g)$ exists in $H_0(m,d)$ then the minimizer exists in $H(m,d)$.

Any function $g \in H(m,d)$ can be written as $g = g_0 + g_1$ where $g_0 \in H_0(m,d)$ and $g_1 \in H_1(m,d)$. Now write I_λ as

$$I_\lambda(g) = \sum_{i=1}^{n} \log\left\{1+exp[g_0(y_i)+g_1(y_i)]\right\}$$
$$- \sum_{i=1}^{n} z_i[g_0(y_i)+g_1(y_i)] + \lambda\|g_1\|^2$$
$$\geq \lambda\|g_1\|^2,$$

so that if $\|g_1\| \to \infty \implies I_\lambda(g) \to \infty$ and by proposition A1.7 this implies that I_λ has a minimum so it will suffice to show that I_λ attains a minimum in

$$H^* = H_0(m,d) \oplus H_1^*(m,d),$$

where

$$H_1^*(m,d) = \left\{g_1 \in H_1(m,d) : \|g_1\| \leq K_1 \quad \text{for some } K_1 > 0\right\}.$$

Let $g \in H^*$, for all $y \in \mathbb{R}^d$ we have

$$|g_1(y)| \leq K_2\|g_1\| \leq K_3. \tag{5.2.2}$$

Again write $g = g_0 + g_1$, where now $g_1 \in H_1^*(m,d)$, then, adding and subtracting $\sum_{i=1}^{n} log\,(1+exp[g_0(y_i)])$ from I_λ and using (5.2.2) we get:

$$I_\lambda(g) \geq \sum_{i=1}^{n} \log\left[\frac{1+exp[g_0(y_i)+g_1(y_i)]}{1+exp[g_0(y_i)]}\right] + I_\lambda(g_0) - nK_3 + \lambda K_3^2. \quad \textbf{(5.2.3)}$$

Looking at an individual term in the summation in (5.2.3) note that

$$g_1(y_i) \geq 0 \Rightarrow \log\left[\frac{1+\exp[g_0(y_i)+g_1(y_i)]}{1+\exp[g_0(y_i)]}\right] \geq 0 \quad (5.2.4)$$

and if $g_1(y_i) < 0$, then

$$|g_1(y_i)| = -g_1(y_i) < K_3$$
$$\Rightarrow g_1(y_i) > -K_3$$

so that

$$\log\left[\frac{1+exp[g_0(y_i)+g_1(y_i)]}{1+exp[g_0(y_i)]}\right] \geq \log\left[\frac{1+exp[g_0(y_i)-K_3]}{1+\exp[g_0(y_i)]}\right]$$
$$= \log\left\{\frac{\exp[-K_3](\exp[K_3]+\exp[g_0(y_i)])}{1+\exp[g_0(y_i)]}\right\} \quad (5.2.5)$$
$$> -K_3$$

then (5.2.4) and (5.2.5) imply that

$$\sum_{i=1}^{n} \log\left\{\frac{1+\exp[g_0(y_i)+g_1(y_i)]}{1+\exp[g_0(y_i)]}\right\} \geq -nK_3. \quad (5.2.6)$$

Hence, by (5.2.3) and (5.2.6) we have that

$$I_\lambda(g) \geq I_\lambda(g_0) + K_4$$
$$\Rightarrow \lim_{\|g\|\to\infty} I_\lambda(g) \geq \lim_{\|g_0+g_1\|\to\infty} I_\lambda(g_0)+K_4$$
$$= \lim_{\|g_0\|\to\infty} I_\lambda(g_0)+K_4 = \infty.$$

since $I_\lambda(g_0)$ attains a unique minimum in $H_0(m,d)$. Therefore, by proposition A1.7 I_λ attains a minimum in $H(m,d)$.

■

Silverman (personal communication) has previously conjectured theorem 5.2.1 and has also noted a rather elegant property of the minimizer of $I_\lambda(g)$: As $\lambda\to\infty$, $\hat{g}_\lambda$ tends to an element of the null space of J_m, so that for $m=2$, the

estimated log-likelihood ratio will be linear and for $m=3$ it will be quadratic. Thus, the parametric estimate for multivariate normals is included as a limiting case. (Compare Wahba 1978 and also Silverman, 1982).

The following theorem characterizes the minimizer of $I_\lambda(g)$ as a Laplacian smoothing spline. The proof of this result is straightforward and follows Wahba and Wendelberger (1982) and section 2.2 of this thesis, and therefore is not given here.

Let $E_m, \vartheta_m, \varphi_1, \ldots, \varphi_M$ be as in section 2.2.

5.2.2 Theorem

The minimizer $\hat{g}_\lambda$ of $I_\lambda(g)$ in $H(m,d)$ if it exists is of the form

$$\hat{g}_\lambda(y) = \sum_{i=1}^{n} \hat{c}_i E_m(y,y_i) + \sum_{j=1}^{M} \hat{d}_j \varphi_j(y),$$

where the vectors $\hat{c} = (c\hat{h}_1, \ldots, \hat{c}_n)^t$ and $\hat{d} = (\hat{d}_1, \ldots, \hat{d}_M)^t$ are the solution to the following non-linear optimization

Problem

Minimize

$$\sum_{i=1}^{n}\left\{\log\left[1+\exp\left[\sum_{j=1}^{n} c_j E_m(y_i,y_j) + \sum_{j=1}^{M} d_j \varphi_j(y_i)\right]\right]\right\} - \sum_{i=1}^{n}\left\{z_i\left[\sum_{j=1}^{n} E_m(y_i,y_j) + \sum_{j=1}^{M} d_j \varphi_j(y_i)\right]\right\} + \lambda c^t E c \qquad (5.2.7)$$

subject to $T^t c = 0$, where now,

$$E := \left[E_m(y_i,y_j)\right] \quad i=1,\ldots,n;\ j=1,\ldots,n$$

and $$T := \left[\varphi_j(y_i)\right] \quad i=1,\ldots,n;\ j=1,\ldots,M.$$

Note that the matrix E above corresponds to the matrix E_{11} of section 2.2 and the matrix T corresponds to the matrix T_1 of the same section.

To find the estimate of g for a given value of λ we must solve a nonlinear optimization problem in $n-M$ variables but there is still the problem of choosing the value of the smoothing parameter.

Since the conditional distribution of Z_i given $Y_i=y_i$ is *binomial*$(1,p)$ where $p = P(Z_i=1 \mid Y_i=y_i)$, then

$$E[Z_i \mid Y_i=y_i] = \frac{\exp[h(y_i)]}{1+\exp[h(y_i)]},$$

so that an ordinary cross-validation estimate of λ would be the value of λ that minimizes

$$\frac{1}{n}\sum_{q=1}^{n}\left\{\exp[\hat{g}_\lambda^{[q]}(y_q)]/\{1+\exp[\hat{g}_\lambda^{[q]}(y_q)]\} - z_q\right\}^2, \qquad (5.2.8)$$

where $\hat{g}_\lambda^{[q]}$ is the estimate of g obtained by minimizing (5.2.7), but leaving out the q^{th} observation. Obviously (5.2.8) would be prohibitive to compute since for each value of λ we must solve n nonlinear programming problems in $n-M$ variables.

Wahba (personal communication) suggested that by minimizing $I_\lambda(g)$ in a subspace of $H(m,d)$ consisting of tensor product B-splines we might be able to get some computational simplifications.

Recently, O'Sullivan (1983) has developed a numerical algorithm together with a generalized cross-validation estimate of λ which is suitable for use with $I_\lambda(g)$.

CHAPTER 6

CONCLUSION

6.1 Summary

We have introduced a nonparametric estimate for the posterior probabilities in the classification problem. The smoothing parameter of the estimate is determined from the data by the method of generalized cross-validation for constrained problems.

Our Monte Carlo experiments attest to the accuracy of the approximate generalized cross-validation function to choose the smoothing parameter. Through this simulation experiments we compared the spline model to estimate posterior probabilities with two parametric models which are very commonly used: the linear model, which is based on the assumption of Normality with the same variance-covariance structure for the populations involved, and the quadratic model which is based on the assumption of Normality with possibly different variance-covariance structures. The linear model performed well only when the samples were generated from Normal distributions with the same variance-covariance matrix. The spline model performed almost as well as the quadratic model with samples generated from Normal distributions with the same or different variance-covariance matrices, and its performance was consistently superior to that of the quadratic model for samples generated from non-Normal distributions.

It can be seen that this method presents a nonparametric alternative to logistic discrimination as well as to survival curve estimation. In logistic regression h(y) is modeled as the logistic function

$$h(y) = \frac{\exp\left[\alpha_0 + \sum_{i=1}^{d} \alpha_i y(i)\right]}{1 + \exp\left[\alpha_0 + \sum_{i=1}^{d} \alpha_i y(i)\right]} \qquad (6.1.1)$$

where $y=(y(1),\ldots,y(d))$ and the parameters $\alpha_0, \ldots, \alpha_d$ are estimated, e.g. by maximum likelihood. See for example, Cox (1966), Day and Kerridge (1967), Anderson (1972), and Anderson and Blair (1982). Here the discriminant functions will be hyperplanes. In survival curve estimation suppose y is a "dose" and $h(y)$ is the probability that a subject survives given a "dose" y. One "observes" that subject i has "dose" y_i, and then one observes a response, which is $z_i=1$ if the subject survives and $z_i=0$ if the subject dies. In logistic survival curve estimation, a function of the form (6.1.1) is fitted to the data (z_i,y_i), however it is clear that the constrained spline estimate provides a nonparametric alternative.

The algorithm developed to estimate the posterior probabilities can be used to solve the more general

Problem 6.1.1

Given $z_i = f(y_i)$, $i=1, \ldots, n$, find $\hat{f}_{n\lambda} \in H(m,d)$ to minimize

$$\sum_{i=1}^{n} \frac{1}{\sigma_i^2}(f(y_i)-z_i)^2 + n\lambda J_m(f)$$

subject to

$$r_j \le f(s_j) \le R_j, \quad j=1, \ldots, k.$$

The subroutines to solve problem 6.1.1 are listed in appendix A2. They allow different measurement error variances, any dimension between 1 and 6 and any value of m satisfying $2m-d > 0$.

Finally, the problem of estimating the likelihood ratio is also considered. A penalized likelihood estimator is given and conditions for existence and uniqueness of such an estimator are given however, no data-based method to choose the smoothing parameter is provided.

6.2 Future work

The results presented in the previous chapters are new and quite promising. There is no doubt that there are many interesting research problems in this area. For example, we believe that convergence rates may be established using the results of Wahba (1979a) and Cox (1982).

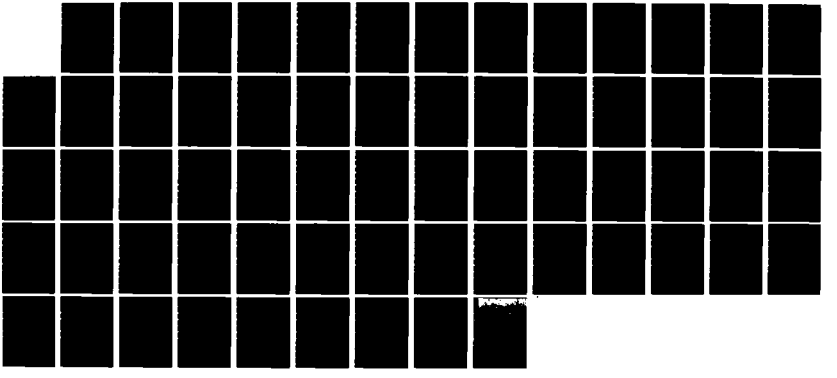
More simulation studies are desirable to study the effect of different sample sizes in the spline estimate.

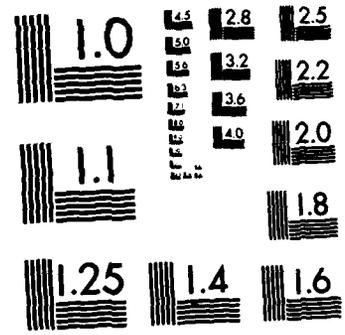
We think that the methods outlined in chapters 2 and 3 can be extended to analyze large data sets following Bates and Wahba (1983).

We plan to incorporate the handling of replicate observations in the algorithm as well as the possibility of enforcing linear constraints not only on the values of the function but also on the values of the derivatives of the function. This would allow, for example, enforcing monotonicity constraints.

AD-A134 518 ESTIMATION OF POSTERIOR PROBABILITIES USING MULTIVARIATE SMOOTHING SPLINE..(U) WISCONSIN UNIV-MADISON DEPT OF STATISTICS M A VILLALOBOS SEP 83 2/2

UNCLASSIFIED UWIS-DS-83-725 ARO-17339.10-MA F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Acknowledgments

I am forever indebted to my wife Rosalinda and my children Miguel and Karla for their love, patience and support throughout my graduate studies.

Special thanks to my mother and sister for their long-distance support and encouragement.

I express my gratitude to my advisor, Professor Grace Wahba for her ideas, guidance, encouragement and understanding. I thank the thesis readers, Sue Leurgans and Douglas Bates and the final exam committee members, David DeMets and Brian Yandell for their helpful comments. Thanks also to the students and staff of the Department of Statistics.

For financial support I thank the Consejo Nacional de Ciencia y Tecnologia - Mexico, and the Department of Statistics of the University of Wisconsin.

APPENDIX A1

SOME MATHEMATICAL OPTIMIZATION RESULTS

In this appendix we present some of the properties of Hilbert spaces and some results in mathematical optimization theory that are used in this thesis. We do not present the proofs of these results because they can be found in several books. The results on Hilbert space can be found in Akhiezer and Glazman (1961) and Aubin (1979). For results on reproducing kernel Hilbert spaces, see Aronszajn (1950). The optimization theoretic results can be found in books such as Luenberger (1969), Ortega and Rheinboldt (1970), and Daniel (1971).

A useful summary of important results relevant to this thesis can be found in the two appendices of Tapia and Thompson (1978). They present proofs of results that cannot be easily found in standard analysis texts.

Review of Hilbert space properties

Throughout this appendix let H denote a Hilbert space of functions that map $\mathbf{R}^d \to \mathbf{R}$. Let H^* be the dual of H, that is, the vector space of all bounded functionals from H into $\mathbf{R}$.

A functional $L: H \to \mathbf{R}$ is said to be bounded if there exists a constant C such that $|Lf| \leq C\|f\|_H$ for all $f \in H$. A functional $L: H \to \mathbf{R}$ is said to be continuous at $f \in H$ if $\{f^n\} \subset H$ and $f_n \to f \Rightarrow Lf^n \to Lf$. L is said to be continuous in $S \subset H$ if it is continuous at each point in S.

A1.1 Proposition

If L is a linear functional defined on H, then the following are equivalent:

(*i*) *L is bounded*
(*ii*) *L is continuous in H*
(*iii*) *L is continuous at one point in H.*

A1.1 Theorem (Riesz representation)

If $L \in H^*$, then there exists a unique $f_L \in H$ such that

$$Lf = \langle f_L, f \rangle \quad \forall f \in H,$$

moreover, the vector space H^* becomes a Hilbert space with inner product

$$\langle L, G \rangle_{H^*} = \langle f_L, f_G \rangle_H \quad \forall L, G \in H^*.$$

A1.1 Definition

A Hilbert space $H(U)$ of functions defined on a set $U \subset \mathbf{R}^d$ is said to be a reproducing kernel Hilbert space (RKHS) if there exists a reproducing kernel $K(\cdot,\cdot)$ functional defined on $U \times U$ such that

$$(i) \quad K(\cdot, t) \in H(U) \quad \forall t \in U$$

and

$$(ii) \quad f(t) = \langle f, K(\cdot, t) \rangle \quad \forall f \in H(U) \text{ and } \forall t \in U.$$

A1.2 Proposition

$H(U)$ is a RKHS if and only if for all $t \in U$ the point evaluation at t is continuous, that is, if and only if there exists C_t such that

$$|f(t)| \leq C_t \|f\| \quad \forall f \in H(U).$$

A1.3 Proposition

Let L be a continuous linear functional defined in the reproducing kernel Hilbert space $H(U)$, and consider the kernel $K(s,t)$, as a function of t, then $LK(s,t) = \eta(s)$ where η is the Riesz representer of L.

Convexity and differential characterizations

A1.2 Definition

Let S be a closed convex subset of H and consider $L: H \to \mathbf{R}$. Then

(i) *L is convex on S* if
$tLf + (1-t)Lg \geq L(tf+(1-t)g) \quad \forall t \in [0,1]$ and $\forall f, g \in S$.

(ii) *L is strictly convex in S* if

$tLf + (1-t)Lg > L(tf+(1-t)g) \quad \forall t \in (0,1)$ and $\forall f, g \in S$, $f \neq g$.

A1.3 Definition

Let $L: H \to \mathbf{R}$. Given $\omega_1, \ldots, \omega_n$, and $f \in H$, by the n^{th} Gateaux variation of L at f in the directions $\omega_1, \ldots, \omega_n$ is given by

$$L^{(n)}f(\omega_1, \ldots, \omega_n) = \lim_{t \to 0} \frac{1}{t}\Big\{L^{(n-1)}(f+t\omega_n)(\omega_1, \ldots, \omega_{n-1}) - L^{(n-1)}(f)(\omega_1, \ldots, \omega_{n-1})\Big\}$$

with $L^{(0)}f = Lf$. If $L^{(1)}f \in H^*$, the Riesz representer of $L^{(1)}f$ is denoted by ∇Lf and is called the gradient of L at f, that is,

$$L^{(1)}(f)(\omega) = \langle \nabla Lf, \omega \rangle \quad \forall\, \omega \in H.$$

A1.4 Proposition

If $\Phi(t) = L(f+t\omega)$ then $\Phi^{(n)}(0) = L^{(n)}(f)(\omega, \ldots, \omega)$.

A1.4 Definition

Let S be a convex subset of H. By the cone tangent to S at f we mean

$$\Omega(f) = \left\{\omega \in H : \exists\, t>0, \textit{ such that } f + t\omega \in S\right\}.$$

Suppose that L is twice Gateaux differentiable in S, then L'' is said to be positive semidefinite relative to S if for each $f \in S$ we have

$$L''(f)(\omega,\omega) \geq 0 \quad \forall\, \omega \in \Omega(f).$$

We say that L'' is positive definite relative to S if

$$L''(f)(\omega,\omega) > 0 \quad \forall\, \omega \in \Omega(f),\ \ \omega \neq 0.$$

A1.5 Proposition

Assume that $L: H \to \mathbf{R}^d$ is twice Gateaux differentiable in a closed convex subset S of H. Then:

(i) L is convex in $S \iff L''$ is positive semidefinite relative to S.

(ii) L is strictly convex in S if L'' is positive definite relative to S.

Optimization problems in Hilbert space

We now consider the existence and uniqueness of a solution to

Problem A1.1

Minimize Lf subject to $f \in S$.

A1.6 Proposition

If L is strictly convex and S is closed and convex, then problem A1.1 has at most one solution.

A1.7 Proposition

Let S be a closed convex subset of H. If $L: S \to \mathbf{R}$ is convex in S, continuous in S and if $\{f^n\} \subset S$ and $\|f^n\| \to \infty \Rightarrow L(f^n) \to \infty$, then, problem A1.1 has at least one solution.

APPENDIX A2

DOCUMENTATION FOR DSCOMP AND DSEVAL

In this appendix we list the documentation for routines DSCOMP which computes the spline with linear constraints and DSEVAL which evaluates the spline at some set of points provided by the user. We also list here all the routines needed to compute the spline, except the routines to solve the quadratic programming problem. These routines are coded in Ratfor (see Kernighan and Plauger, 1976).

```
*****************************************************************************
*                                  DSCOMP                                   *
*****************************************************************************

subroutine dscomp(x,ldx,dim,nobs,s,lds,ncons,z,m,bl,bu,isigma,
                  sigma,bigeig,zero,finity,indlam,lamvec,nvalle,
                  nvalri,nvalam,isave,istart,istate,nactiv,itmax,
                  ioptv,iqpout,iactch,wmach,powers,ldp,bigm,coef,
                  hhat,voflam,thetam,iwork,liwork,work,lwork,ierr)

double precision x(ldx,1),s(lds,1),z(1),finity,zero,bigeig,sigma(1),
                 bl(1),bu(1),coef(1),hhat(1),wmach(15),voflam(1),
                 bigeig,lamvec(nvalle+nvalri+1),thetam,work(1)

integer ldx,lds,m,dim,istate(1),nobs,ncons,lwork,istart,isigma,
        liwork,iwork(1),itmax,nvalam,istate(1),nactiv,isave,
        ioptv,nvalle,nvalri,iactch(1),indlam,powers(ldp,1),ldp,bigm
#
implicit double precision (a-h,o-z)
# This routine computes a thin plate spline with linear constraints
# as the solution to the following problem
#
# minimize:
#
#  nobs                  2           2
#  SUM  (z(i) - h(x(i)) /(sigma(i))  +  lambda * J (h)
#  i=1                                             m
#
# subject to
#
#               bl(1)     .le.   h(s(1))    .le. bu(1)
#               bl(2)     .le.   h(s(2))    .le. bu(2)
#                                    .
#                                    .
#                                    .
#               bl(ncons) .le. h(s(ncons)) .le. bu(ncons)
#
#
#       x       Double(ldx,nobs).
#               Array containing the abscissas of the points at
#               which observations are. The rows of this matrix
#               correspond to the dimension of the space, and
#               the columns to the number of observations.
#               For example x(2,100) is the second coordinate
#               of the 100th point.
#
#       ldx     Integer constant or variable.
#               On input ldx must contain the leading dimension
#               of array x as defined in the dimension statement
#               in the main program. ldx should be greater than
#               or equal than the dimension of the space.
```

```
#
#       dim     Integer constant or variable.
#               Dimension of the space.
#
#       nobs    Integer constant or variable.
#               Contains the number of observations for this problem.
#
#       s       Double (lds,ncons)
#               Array containing the points at which the linear
#               constraints are enforced. For example, s(2,100)
#               denotes the second coordinate of the 50th point
#               where constraints are enforced.
#
#       lds     Integer variable or constant.
#               Leading dimension of the array s. lds should be
#               greater than or equal to the dimension of
#               the space.
#
#       ncons   Integer variable or constant.
#               Number of linear constraints being enforced.
#
#       z       Double (nobs).
#               Vector of observations at the points in x.
#
#       m       Integer variable or constant.
#               Degree of smoothing for the spline. If the
#               dimension of the space,dim is smaller than 4
#               then 2 .le. m .le. 7.  If dim= 4 then m = 3, 4
#               or 5.  If dim= 5, m = 4 or 5. If dim= 6, m=4.
#
#       bl      Double (nobs + 2*ncons).
#               Vector containing the lower bounds for the
#               constrained spline in the last ncons positions.
#               The first nobs+ncons positions are used by the
#               quadratic programming routine.
#
#       bu      Double (nobs + 2*ncons).
#               Vector containing the upper bounds for the
#               constrained spline in the last ncons positions.
#               The first nobs+ncons positions are used by the
#               quadratic programming routine.
#
#       isigma  Integer variable or constant.
#               Indicates whether or not relative weights for
#               approximate errors on the observed values are
#               to be specified. See sigma below.
#
#               isigma = 1 - relative weigths are not specified
#                            and assumed to be all equal to 1.
#               isigma = 0 - relative weithts are specified in
#                            sigma.
```

```
#
#               If isigma different from 1 and 0, isigma = 1 is
#               assumed.
#
#       sigma   Double (nobs).
#               Array of size at least nobs specifying the relative
#               weights for approximate errors on the observed
#               values.  If sigma = 1 then sigma is not used and
#               need not be dimensioned in the calling program.
#
#       bigeig  Double precision variable.
#               If lambda for unconstrained problem is infinity,
#               bigeig should contain the largest eigenvalue
#               obtained when the unconstrained problem was solved
#               If lambda for the unconstrained problem is finite
#               bigeig is not referenced.
#
#       indlam  Integer variable.
#               On return, indlam will contain a number such
#               that lamvec(indlam) = log of the estimate of the
#               smoothing parameter lambda.
#
#       lamvec  Double (nvalle + nvalri + 1).
#               On entry: lamvec (1) should contain the estimate of
#                         lambda that minimizes the generalized
#                         cross-validation function for the uncons-
#                         trained problem. If this value is infinity
#                         then lamvec(1) should be set to -1.d0.
#              On return:lamvec will contain a grid of values in
#                         units of log of lambda where the generalized
#                         cross-validation function for constrained
#                         problems was evaluated. Also, lamvec(indlam)
#                         will contain the log of the value of lambda
#                         that minimized the GCVC function.
#
#       nvalle  Integer variable.
#               On entry nvalle should contain the number of values
#               smaller than lamvec(1) at which GCVC is to be evaluated.
#
#       nvalri  Integer variable.
#               On entry nvalri shoud contain the number of values
#               larger than lamvec(1) at which GCVC is to be evaluated.
#
#               If both nvalle and nvalri are less than or equal to
#               zero then nvalri and nvalle are set to 15.
#
#               If lamvec(1) = 0.d0 then nvalle should be equal to zero.
#               If lamvec(1) = -1.d0, that is the estimate of lambda
#                              from the unconstrained problem is
#                              infinity, then nvalri should be zero.
#
```

```
#       nvalam  Integer variable.
#               On return nvalam will contain the number of values of
#               lambda at which V(lambda) was evaluated.
#
#       isave   Integer variable or constant.
#               Specifies whether or not various intermediate results
#               computed by dscomp are to be saved or restored from
#               file.  If two or more problems are to be solved with
#               the same dim, m, x, s, and sigma, i. e., changes only
#               in z, some of the computations can be bypass  for
#               the second and subsequent problems by savin  termediate
#               results computed for the first problem.
#
#               isave = 1 - Do not use any intermediate resu   saved
#                           and save results computed on fil
#
#               isave =-1 - Use intermediate results saved o.  ..le.
#
#               isave different from -1 and 1 - Do not use any intermi-
#                           diate results saved and do not save any
#                           results computed on file.
#
#       istart  Integer variable or constant.
#               Indicates whether or not an initial guess for the points
#               at which the the constraints will be active for the
#               value of lambda that minimizes GCVC will be given.
#
#               istart = 1 - Guess for set of active constraints is
#                           specified in the vector istate (see bellow).
#
#               istart different from 1 - No initial guess is given.
#
#               It is recommended to use istart = 1 and use the set of
#               points where the constraints are most seriously violated
#               as the initial guess (see istate).
#
#       istate  Integer (nobs + 2*ncons).
#               On entry: The last ncons positions of this array
#                         indicate the initial guess for the set of
#                         active constraints.
#
#               On return:The last ncons positions of this array indicate
#                         the set of active constraints at the solution.
#
#               istate (nobs+i) = 0 - ith. linear constraint not active.
#
#               istate (nobs+i) = 1 - ith. linear constraint active at
#                                     its lower bound.
#
#               istate (nobs+i) = 2 - ith. linear constraint active at
#                                     its upper bound.
```

```
#
#       nactiv  Integer variable.
#               On entry:  The number of active constraints in the initial
#                          guess (if istart=1).
#
#               On return: The number of active constraints at the
#                          solution.
#
#       itmax   Integer variable.
#               Maximum number of iterations for the quadratic
#               programming problem. If itmax less than or equal zero
#               then itmax is set to 100.
#
#       ioptv   Integer variable or constant.
#               Indicates whether or not optimization of the generalized
#               cross-validation function for constrained problems is
#               desired.
#
#               ioptv = 1 - Optimization of V(lambda) is desired.
#
#               ioptv .ne. 1 - Optimization of V(lambda) not desired.
#
#       iqpout  Integer variable or constant.
#               Controls printing for quadratic programming routine.
#               iqpout = 0 - Nothing is printed by qpfc.
#                      = 2 - One line of output for each quadratic
#                            programming problem solved.
#                      >10 - Almost all relevant information is
#                            printed for debugging purposes.
#
#       iactch  Integer (nvalle + nvalri + 1).
#               If optimization of V(lambda) is requested then this
#               array indicates for each value of lambda tried whether
#               or not the set of active constraints changed and whether
#               or not the quadratic programming problem for each value
#               of lambda had a solution.  If for a specific value of
#               lambda, say, lamvec(j) the quadratic programming problem
#               did not have a solution then iactch is set to:
#
#               inform = 2 - The problem appears to be infeasible.
#
#               inform = 4 - Too many iterations have been perfomed.
#
#               inform = 5 - Too many iterations have been performed
#                            without changing the solution. The
#                            degeneracy is unresolved.
#
#               inform = 6 - An active constraint has become infeasible
#                            The constraints are likely to be very badly
#                            scaled. Try a different starting point.
#
```

If lamvec (j) is not any of the above numbers then lamvec (j) will be either zero or one. When, say lamvec (j) = 1 and lamvec (j+1) = 1, this means that changing from the jth. value of lambda to the next one did not change the set of active constraints. If lamvec(j)=1 and lamvec(j+1)=0, then the active set of constraints changed.

wmach Double (15).
Machine dependent constants required by the quadratic programming routine qpfc.

wmach(1) - Base of floating point arithmetic
wmach(2) - No. of base wmach(1) digits of precision.
wmach(3) - Floating point precision.
wmach(4) - sqrt(wmach(3)).
wmach(5) - Smallest positive floating point number.
wmach(6) - Sqrt(wmach(5)).
wmach(7) - Largest positive floating point number.
wmach(9) - Sqrt (wmach(7)).
wmach(10) - Standard file number for the input stream.
wmach(11) - Standard file number for the output stream.

powers Integer (ldp,bigm).
On return powers(i,j) contains the power at which the ith. component in dim-space is raised, corresponding to the jth. d-coefficient.

For example, if m=3 and d=2, powers would be given by:

```
0  0  0  1  1  2
0  1  2  0  1  0
```

ldp Integer variable or constant.
Specifies leading dimension of the array powers as defined in the dimension statement of the calling program. ldp should be larger than or equal to dim.

bigm Integer variable.
On return bigm will contain the number of d-coefficients of the spline.

$$bigm = \frac{(dim + m - 1)!}{(dim! * (m-1)!)} .$$

coef Double (nobs + ncons + bigm).
On return: coef will contain the coefficients of the spline. The coefficients will be arranged as follows:

| c |

```
#                       coef = | b |
#                              | d |
#
#                       where c is an nobs dimensional vector that
#                       will contain the coefficients of the spline
#                       corresponding to the points where the obser-
#                       vations were taken. b is an ncons dimensional
#                       vector containing the coefficients of the
#                       spline corresponding to the points where the
#                       constraints are enforced. d is a bigm
#                       dimensional vector containing the coefficients
#                       of the polynomial part of the spline.
#
#       hhat    Double (nobs).
#               On return, hhat will contain the estimate of the function
#               h at the sample points x.
#
#       voflam  Double (nvalle+nvalri+1).
#               If optimization of V(lambda) was requested voflam
#               will contain the value of V(lamda) at the values of
#               lambda in lamvec.
#               A negative value of voflam(j) means that the quadratic
#               problem could not be solved for the lambda=lamvec(j),
#               see also iactch.
#
#       thetam  Double precision variable.
#               On return thetam will contain the constant that
#               multiplies E .
#                          m
#
#       iwork   Integer (liwork).
#               Integer work array of dimension at least liwork.
#
#       liwork  Integer variable.
#               On entry iwork specifies the dimension of the array
#               iwork.  liwork should be greater than or equal to
#               ntotal + 2*ncons.
#
#       work    Double (lwork).
#               Double precision work array of dimension lwork.
#
#       lwork   Integer variable or constant.
#               On entry lwork specifies the dimension of work.
#
#               lwork .ge. ntotal*(2*ntotal+2*bigm+nobs+ncons+2)+
#                          nmbigm*(nmbigm+2)+maximum(nnn1,nnn2)
#
#               where ntotal=nobs+ncons
#                     nmbigm=ntotal-bigm
#                     nnn1=ntotal*(ntotal+ncons+10) +
#                          ncons*(ncons+5) + 2
```

```
#                    nnn2=ntotal*(3*ntotal+1)
#
#       ierr    Integer variable.
#               On return ierr will contain an error code.
#
#               ierr=0 - No errors occurred.
#
#               ierr>0 - Error(s) in input parameters occured.
#                        ierr is a number of the form
#                        abcdefg, where each digit is either
#                        zero [no error occured] or one
#                        [some parameter(s) value(s) is (are)
#                        invalid], as follows:
#
#                        a=1 - Invalid combination of m and dim.
#                        b=1 - Ncons < 1.
#                        c=1 - nobs+ncons-bigm < 1.
#                        d=1 - Either ldx<dim or lds<dim or ldp<dim.
#                        e=1 - sigma(i) .le. 0 for some i.
#                        f=1 - lwork smaller than required.
#                        g=1 - liwork smaller than required.
#
#               ierr<0 - Errors in the solution of the problem
#                        independent of the quadratic programming
#                        problem occured:
#
#                        ierr=-10 - Values for nobs,ncons,bigm in
#                                   first record of matrix-file do
#                                   not agree with current values.
#
#                        ierr=-11 - Insufficient data in matrix-file.
#
#                        ierr=-20 - Matrix b=Q ' * E(1) * Q
#                                                 2          2
#                                   is not positive definite. This
#                                   may be caused by replicates in
#                                   the original data set.
#
ierr=0
#
call dsbigm(dim,m,bigm)
#
#                        compute thetam.
#
if (bigm != 0) call dsteta(dim,m,thetam)
ntotal=nobs+ncons
nmbigm=ntotal-bigm
#
#                        allocate work storage
#
n1=1
```

```
n2=ntotal*bigm+n1
n3=ntotal*bigm+n2
n4=bigm+n3
n5=ntotal*ntotal+n4
n6=bigm+n5
n7=ntotal*ntotal+n6
n8=nmbigm*nmbigm+n7
n9=nobs*ntotal+n8
n10=ncons*ntotal+n9
n11=ntotal+n10
n12=ntotal+n11
nnn1=ntotal*(ntotal+ncons+10)+ncons*(ncons+5)+2
nnn2=ntotal*(3*ntotal+1)
lwk=max0(nnn1,nnn2)
liwk=ntotal+2*ncons
#
#
#                          minimum dimension of work
#
minwk=n12+lwk
iflag=0
if (isigma != 0) isigma=1
if (isigma != 1) {
   do i=1,nobs
      if(sigma(i) <= 0.d0)
         iflag=1
}
if (liwork < liwk) ierr=1
if (lwork < minwk) ierr=ierr+10
if (iflag == 1)    ierr=ierr+100
if (ldx < dim | lds < dim | ldp < dim) ierr=ierr+1000
if (nmbigm <= 0)   ierr=ierr+10000
if (ncons < 1)     ierr=ierr+100000
if (bigm == 0)     ierr=ierr+1000000
if (ierr > 0) return
if (itmax <= 0) itmax=100
if (ioptv != 1) ioptv=0
if (iabs(isave) != 1) isave=0
if (istart != 1) istart = 0
if (nvalri+nvalle <= 0) {
   if (lamvec(1) < -0.5d0){
      nvalri=0
      nvalle=30
   }
   else {
      if (lamvec(1) <= 0.d0) {
         nvalri=30
         nvalle=0
      }
      else {
         nvalri=15
```

```
            nvalle=15
         }
      }
   }
#
#                        set machine constants
#
call dsmach(wmach)
zero=wmach(5)
finity=wmach(9)
nvalam=nvalle+nvalri+1
call dscom2(x,ldx,dim,nobs,s,lds,ncons,z,m,bl,bu,isigma,sigma,
           bigeig,zero,finity,indlam,lamvec,nvalle,nvalri,
           nvalam,isave,istart,istate,nactiv,itmax,ioptv,iqpout,
           iactch,wmach,powers,ldp,nactiv,coef,hhat,
           voflam,thetam,ntotal,nmbigm,bigm,work(n1),work(n2),
           work(n3),work(n4),work(n5),work(n6),work(n7),
           work(n8),work(n9),work(n10),work(n11),work(n12),
           lwk,iwork,liwork,ierr)
ierr=-ierr
return
end
```

```
*****************************************************************
*                            DSEVAL                             *
*****************************************************************

subroutine dseval(x,ldx,dim,m,nobs,s,lds,ncons,xtab,ldxtab,
                  ntab,coef,powers,ldp,hofxta,ierr)
double precision x(ldx,1),s(lds,1),xtab(ldxtab,1),coef(1),
                 hofxta(1),p(6),hxta
integer dim,nobs,ldx,lds,ldxtab,ntab,powers(ldp,1),ierr,m,bigm
#
# This routine evaluates the spline h, computed by dscomp
# at the set ntab points in xtab.  The resulting values of
# the spline are put in hofxta.
#
#       x       Double(ldx,nobs).
#               Array containing the abscissas of the points at
#               which observations are. The rows of this matrix
#               correspond to the dimension of the space, and the
#               columns to the number of observations. For example
#               x(2,9) is the second coordinate of the 9th point.
#
#       ldx     Integer constant or variable.
#               On input ldx must contain the leading dimension of
#               array x as defined in the dimension statement in
#               the main program. ldx should be greater than or
#               equal to the dimension of the space.
#
#       dim     Integer constant or variable.
#               Dimension of the space.
#
#       nobs    Integer constant or variable.
#               Contains the number of observations for this problem.
#
#       s       Double (lds,ncons)
#               Array containing the points at which the linear
#               constraints are enforced. For example, s(2,100)
#               denotes the second coordinate of the 50th point
#               where constraints are enforced.
#
#       lds     Integer variable or constant.
#               Leading dimension of the array s. lds should
#               be greater than or equal to the dimension of
#               the space.
#
#       ncons   Integer variable or constant.
#               Number of linear constraints being enforced.
#
#       m       Integer variable or constant.
#               Degree of smoothing for the spline. If the
#               dimension of the space,dimis smaller than 4
#               then 2 .le. m .le. 7.  Ifdim= 4 then m =
```

3, 4 or 5. Ifdim= 5, m = 4 or 5. Ifdim= 6
m = 4.

powers Integer (ldp,bigm).
On entry powers(i,j) should contain the power at which the ith. component in dim-space is raised, corresponding to the jth. d-coefficient. This matrix is computed by dscomp.

ldp Integer variable or constant.
Specifies leading dimension of the array powers as defined in the dimension statement of the calling program. ldp should be larger than or equal to dim.

bigm Integer variable.
On return bigm will contain the number of d-coefficients of the spline.

$$bigm = \frac{(dim + m - 1)!}{(dim! * (m-1)!)} .$$

coef Double (nobs + ncons + bigm).
On entry coef should contain the coefficients of the spline as obtained by dscomp. The coefficients should be arranged as follows:

$$coef = \begin{vmatrix} c \\ b \\ d \end{vmatrix}$$

where c is an nobs dimensional vector that will contain the coefficients of the spline corresponding to the points where the observations were taken. b is an ncons dimensional vector containing the coefficients of the spline corresponding to the points where the constraints are enforced. d is a bigm dimensional vector containing the coefficientsof the polynomial part of the spline.

hofxta Double (ntab).
On return, hofxta will contain the estimate of the function h at the ntab points in xtab.

thetam Double precision variable.
On return thetam will contain the constant that multiplies E_m.

```
#
#
#       ierr    Integer variable.
#               On return ierr will contain an error code.
#
#               ierr=0 - No errors occurred.
#
#               ierr>0 - Error(s) in input parameters occured.
#                        ierr is a number of the form
#                        abc, where each digit is either
#                        zero [no error occured] or one
#                        [some parameter(s) value(s) is (are)
#                        invalid], as follows:
#
#                        a=1 - Either ldx<dim or lds<dim or ldp<dim
#                              or ldxtab < dim.
#                        b=1 - nobs+ncons-bigm < 1.
#                        c=1 - Invalid combination of m and dim.
#
#
#                        Get bigm
#
call dsbigm(dim,m,bigm)
#
#                        Get thetam
#
if (bigm != 0) call dsteta(dim,m,thetam)
ntotal=nobs+ncons
nmbigm=ntotal-bigm
if (ldx<dim | lds<dim | ldxtab<dim | ldp<dim) ierr = 1
if (nmbigm <= 0) ierr = ierr+10
if (bigm == 0) ierr=ierr+100
if (ierr != 0) return
do j=1,ntab {
   do i=1,dim{
      p(i)=xtab(i,j)
   }
#                        Evaluate spline at point p
   call dshofp(p,x,ldx,dim,nobs,s,lds,ncons,ntotal,m,thetam,
               bigm,powers,ldp,coef,hxta)
   hofxta(j)=hxta
}
return
end
```

```
*****************************************************************
*                          DSBIGM                               *
*****************************************************************

subroutine dsbigm (dim, m, bigm)
#
#     get bigm for dim,m if legal.  if illegal, bigm
#     is set to zero.  legal values of dim and m are:
#
#     m  dim  1    2    3    4    5    6
#     2       yes  yes  yes  no   no   no
#     3       yes  yes  yes  yes  yes  no
#     4       yes  yes  yes  yes  yes  yes
#     5       yes  yes  yes  yes  no   no
#     6       yes  yes  yes  no   no   no
#     7       yes  yes  yes  no   no   no
#
#
implicit integer (a-z)
dimension mtab(36)
data mtab / 2,  3,  4,  5,  6,  7,
            3,  6, 10, 15, 21, 28,
            4, 10, 20, 35, 56, 84,
            0, 15, 35, 70,  0,  0,
            0, 21, 56,  0,  0,  0,
            0,  0, 84,  0,  0,  0 /
#
if (dim .lt. 1 .or. dim .gt. 6) go to 13
if (m .lt. 2 .or. m .gt. 7) go to 13
l = 6*dim + m - 7
bigm = mtab(l)
return
#     error return
13 bigm = 0
return
end


*****************************************************************
*                          DSCBD                                *
*****************************************************************

subroutine dscbd(ntotal,nobs,nmbigm,bigm,coef,tsigqr,
                 qrtsia,work,isigma,sigma)
double precision coef(ntotal),tsigqr(ntotal,bigm),qrtsia(bigm),
                 work(1),sigma(nobs)
integer ntotal,nobs,nmbigm,bigm,isigma
#
# Obtain vector of coefficients of the spline
#
#              | c |
```

```
#       coef = | b |
#              | d |
# Recall that the vector coef contains the solution
# to the lower dimensional problem (ntotal - bigm)
#
#                       Zeroes to first bigm pos. of work
#
do i=1,bigm
   work(i)=0.d0
#                       Copy coef into work(bigm+1)
#
call dcopy(ntotal,coef,1,work(bigm+1),1)
#
#                       get
#
#                       | c |
#                       |   | = Q   * coef = |Q    Q  |*work
#                       | b |    2           | 1    2 |
#
job  = 10000
call dqrsl(tsigqr,ntotal,ntotal,bigm,qrtsia,work,coef,
          dmy,dmy,dmy,dmy,job,info)
#
#                       Copy last bigm elements of work into
#                       last elements of coef
#
call dcopy(bigm,work(ntotal+1),1,coef(ntotal+1),1)
#
#                       Rescale by sigma if sigmai's were
#                       read
#
if (isigma == 1)
   return
else {
   do i=1,nobs
      coef(i)=coef(i)/sigma(i)
}
return
end


*************************************************************************
*                              DSCOLE                                   *
*************************************************************************

subroutine dscole(j,cje,ntotal,x,ldx,nobs,s,lds,ncons,m,pi,pj,
                  dim,thetam)
double precision x(ldx,1),s(lds,1),pi(dim),pj(dim),thetam,cje(1)
integer j,ntotal,ldx,lds,dim
#
# Compute lower diagonal elements of the jth column of E
```

```
# and put in vector cje.
#
#              If j <= nobs retrieve jth sample point
#              else retrieve the (j-nobs) grid point in s
#
if (j <= nobs) {
   do i=1,dim
      pj(i)=x(i,j)
}
else {
   do i=1,dim
      pj(i)=s(i,j-nobs)
}
if (j<=nobs)
   jl=1
else
   jl=j-nobs
do i=j,nobs {
   do il=1,dim
      pi(il)=x(il,i)
#                          Obtain E(pi,pj)
   call dseij(pi,pj,dim,m,cje(i))
   cje(i)=cje(i)*thetam
}
do i=jl,ncons {
   do il=1,dim
      pi(il)=s(il,i)
   il=i+nobs
   call dseij(pi,pj,dim,m,cje(il))
   cje(il)=cje(il)*thetam
}
return
end


**************************************************************************
*                                DSCOM2                                  *
**************************************************************************

subroutine dscom2(x,ldx,dim,nobs,s,lds,ncons,z,m,bl,bu,isigma,sigma,
          bigeig,zero,finity,indlam,lamvec,nvalle,nvalri,
          nvalam,isave,istart,istate,nactiv,itmax,ioptv,iqpout,
          iactch,wmach,powers,ldp,nactiv,coef,hhat,
          voflam,thetam,ntotal,nmbigm,bigm,tsigma,tsigqr,
          qrtsia,esigma,qrtsil,hessl,hess2,qplima,qpcons,qplive,
          eigen,work,lwork,iwork,liwork,ierr)
#
double precision x(ldx,nobs),s(lds,ncons),finity,zero,bigeig,
                 sigma(1),bl(1),bu(1),coef(1),hhat(1), voflam(1),
                 thetam,tsigma(ntotal,1),tsigqr(ntotal,1),qrtsia(1),
                 esigma(ntotal,1),qrtsil(1),hessl(ntotal,1),
```

```
                hess2(nmbigm,1),qplima(nobs,1),
                qpcons(ncons,1),qplive(1),eigen(1),
                work(1)
#
integer ldx,lds,m,dim,istate(1),nobs,ncons,lwork,
        liwork,iwork(1),itmax,istart,isigma,isave,
        ioptv,nvalle,nvalri,istate(1),nactiv,
        powers(ldp,1),ldp,bigm,iactch(1),
        indlam,ntotal,nmbigm
#
logical start
#
#
# Second level routine to compute a thin plate spline with linear
# constraints as the solution to the following problem
#
# minimize:
#
#  nobs                2           2
#  SUM  (z(i) - h(x(i)) /(sigma(i))  + lambda * J (h)
#  i=1                                           m
#
# subject to
#
#              bl(1)     .le.   h(s(1))    .le. bu(1)
#              bl(2)     .le.   h(s(2))    .le. bu(2)
#                                  .
#                                  .
#                                  .
#              bl(ncons) .le. h(s(ncons)) .le. bu(ncons)
#
#
# For a description of the variables and arrays see comments in
# dscomp.r
#
#
#                       Get powers of polynomials
#
nt=ntotal
nmb=nmbigm
nprin1=min0(20,ntotal)
nprin2=min0(10,nobs)
nprin3=min0(10,ncons)
call dspoly (dim,m,powers,ldp)
#
#                       Scale z data by 1/sigma if necesary
#
#
if (isigma == 0) call dszdis (z,nobs,sigma)
#
#                       If matrices previously saved read them,
```

```
#                       skip computations and go to 100
#
if (isave == -1) {
   open (1,file='splmatrix',form='unformatted')
   rewind (1)
   call dsrdl(ntotal,nobs,ncons,bigm,dim,nmbigm,hess1,hess2,qplima,
              qpcons,tsigqr,qrtsia,esigma,ierr)
   if (ierr != 0) return
   go to 100
}
#
#                       Form tsigma
#
call dstsig(x,ldx,s,lds,powers,ldp,tsigma,ntotal,bigm,nobs,ncons,
            dim,isigma,sigma)
#
#                       Copy tsigma in tsigqr
#
do j=1,bigm
   call dcopy (ntotal,tsigma(1,j),1,tsigqr(1,j),1)
#
#                       Form QR decomposition of tsigma
#
job = 0
call dqrdc(tsigqr,ntotal,ntotal,bigm,qrtsia,jpvt,work,job)
#
#                       Form esigma
#
call dsesig(esigma,ntotal,nobs,ncons,isigma,sigma,x,ldx,
            s,lds,work,thetam,dim,m)
#
#                       Form hess1
#
call dshes1(hess1,ntotal,nmbigm,nobs,tsigqr,qrtsia,tsigma,bigm,
            esigma,work)
#
#                       Form hess2 = Q ' * E * Q
#                                     2        2
#
call dsqtaq(tsigqr,ntotal,ntotal,bigm,qrtsia,esigma,ntotal,
            hess2,nmbigm,work(1),work(ntotal+1))
#
#                       Compute qpcons (matrix of linear const.)
#
call dscons(qpcons,ncons,ntotal,nobs,tsigma,bigm,tsigqr,qrtsia,
            nmbigm,esigma,work)
#
#                       Compute qplima (matrix that post-multiplies
#                       zsigma to obtain linear term in quadratic
#                       function.  Also, hhat is obtained by
#                       multiplying qplima*coef
```

```
#
call dslima(qplima,nobs,ntotal,bigm,nmbigm,tsigma,tsigqr,
           qrtsia,esigma,work)
#
#                       Write intermediate results if required
#
if (isave == 1) {
   open (1,file='splmatrix',form='unformatted')
   rewind (1)
   call dswrtl(ntotal,nobs,ncons,bigm,dim,nmbigm,hessl,
               hess2,qplima,qpcons,tsigqr,qrtsia,esigma)
}
#
100 continue
#
#                       Compute qplive=vector defining linear term
#
call dslive(qplive,ntotal,z,nobs,qplima)
#
#                       Find coefficients of spline doing optimization
#                       with respect to lambda if necessary
#
if (istart == 1)
   start = .true.
else
   start = .false.
nnn1=ntotal*(ntotal+ncons+9)+ncons*(ncons+3)+1
nnn2=ntotal*(ncons+3)+1
lwk=max0(nnn1,nnn2)
liwk=ntotal+ncons
call dssolv(hessl,ntotal,hess2,nmbigm,bigm,qplima,qplive,qpcons,ncons,
           nobs,tsigma,qrtsil,lamvec,voflam,indlam,coef,ioptv,iqpout,
           nvalle,nvalri,nvalam,zero,finity,bl,bu,start,istate,iwork,
           iwork(ncons+1),iactch,wmach,bigeig,itmax,hhat,z,esigma,
           nactiv,eigen,work,ierr)
if (ierr != 0) return
#
#                       obtain final coefficients of spline
#                       and put again in vector coef in the
#                       form:
#                                 | c |
#                       coef =    | b |
#                                 | d |
#
call dscbd(ntotal,nobs,nmbigm,bigm,coef,tsigqr,qrtsia,work,isigma,sigma)
#
#                       Multiply z values by sigma if sigma was read
#
if (isigma == 0) call dszmus(z,nobs,sigma)
return
end
```

```
*****************************************************************
*                            DSCONS                             *
*****************************************************************

subroutine dscons(qpcons,ncons,ntotal,nobs,
                  tsigma,bigm,tsigqr,qrtsia,nmbigm,esigma,work)
double precision qpcons(ncons,ntotal),tsigma(ntotal,bigm),
                 tsigqr(ntotal,bigm),qrtsia(bigm),
                 esigma(ntotal,ntotal),work(1)
integer ncons,ntotal,nobs,bigm,nmbigm
#
# Compute matrix qpcons which defines linear constraints
#
#                |    |E   | |
#                |Q '*| 12 | |
#                | 2  |E   | |
#     qpcons =   |    | 22 | |
#                |            |
#                |   T '      |
#                |    2       |
#
job=01000
i2=nmbigm+1
i3=bigm+1
do i=1,ncons {
   i1=nobs+i
#
#                        Compute
#                                |Q '|
#                                | 1 |
#                        work =  |   | * ilst. col. of esigma
#                                |Q '|
#                                | 2 |
#
#
   call dqrsl(tsigqr,ntotal,ntotal,bigm,qrtsia,
              esigma(1,i1),dmy,work(1),dmy,dmy,dmy,
              job,info)
#
#                        Copy last nmbigm elements of work into
#                        first nmbigm columns of ith. row of qpcons
#
   call dcopy(nmbigm,work(i3),1,qpcons(i,1),ncons)
#
#                        Copy ilst row of tsigma into last
#                        bigm columns of ith row of qpcons
#
   call dcopy(bigm,tsigma(i1,1),ntotal,qpcons(i,i2),ncons)
}
return
end
```

```
**********************************************************************
*                              DSEIGE                                *
**********************************************************************

subroutine dseige(ntlmm,a,b,eigen,work,ierr)
double precision a(ntlmm,ntlmm),b(ntlmm,ntlmm),eigen(1),work(1)
integer ntlmm,ierr
#
# Solve generalized eigenvalue problem
#
#       a*PHI  = EIG  * b * PHI
#            v                v
#
# to get the neig positive eigenvalues of
#        -1
#       b  *a
#
ierr=0
n1=1
n2=ntlmm+1
call reduc(ntlmm,ntlmm,a,b,work(n2),ierr)
if (ierr == 7*ntlmm+1)
#                         b is not positive definite
   return
call tredl(ntlmm,ntlmm,a,eigen,work(n1),work(n2))
call tqlrat(ntlmm,eigen,work(n2),ierr)
if (ierr > 0) {
   iiii=ierr-1
   print *,' '
   print *,'****** In routine dseige, only eigenvalues 1 to ',iiii
   print *,'       are correct'
   for (i=ierr;i<=ntlmm;i=i+1)
      eigen(i)=0.0d0
}
ierr=0
return
end


**********************************************************************
*                              DSEIJ                                 *
**********************************************************************

subroutine dseij(pi,pj,dim,m,eij)
double precision pi(dim),pj(dim),eij
integer dim,m
implicit double precision (a-h,o-z)
#
# Compute
#             |     |2m-dim   |     |
# e(pi,pj) = |p -p |        ln|p -p |   if dim is even,
```

```
#            | i  j|          | i  j|
#
#            |     |2m-dim
#          = |p -p |                    if dim is odd.
#            | i  j|
#
go to (1,2,3,4,5,4),dim
#
1 mu=2*m-1
  s=dabs(pi(1)-pj(1))
  eij=s**mu
  return
2 mu=m-1
  s=(pi(1)-pj(1))**2 + (pi(2)-pj(2))**2
  if (mu == 1)
     eij=0.5d0*dlog(s)*s
  else
     eij=0.5d0*dlog(s)*s**mu
  return
3 mu=2*m-3
  s=(pi(1)-pj(1))**2+ (pi(2)-pj(2))**2+ (pi(3)-pj(3))**2
  if (mu == 1)
     eij=dsqrt(s)
  else
     eij=dsqrt(s)**mu
  return
4 mu=m-dim/2
  s=(pi(1)-pj(1))**2
  do i=2,dim
     s=s+(pi(i)-pj(i))**2
  if (mu == 1)
     eij=0.5d0*dlog(s)*s
  else
     eij=0.5d0*dlog(s)*s**mu
  return
5 mu=2*m-5
  s=(pi(1)-pj(1))**2
  do i=2,dim
     s=s+(pi(i)-pj(i))**2
  if (mu == 1)
     eij=dsqrt(s)
  else
     eij=dsqrt(s)**mu
  return
end
```

```
**********************************************************************
*                              DSESIG                                *
**********************************************************************

subroutine dsesig(esigma,ntotal,nobs,ncons,isigma,sigma,
                  x,ldx,s,lds,work,thetam,dim,m)
implicit double precision (a-h,o-z)
double precision esigma(ntotal,ntotal),sigma(nobs),x(ldx,1),
                 s(lds,1),work(1),thetam
integer ntotal,nobs,ncons,isigma,ldx,lds,dim
#
# Compute:
#                |     -1          -1          -1     |
#                |SIGMA   E   SIGMA       SIGMA  E   |
#                |         11                     12 |
# esigma   =     |                                   |
#                |              -1                   |
#                |     E   SIGMA             E       |
#                |      21                    22     |
#
#
do j=1,ntotal {
#                       Get lower diagonal elements of jth. column of E
#
#
   nw1=dim+1
   call dscole(j,esigma(1,j),ntotal,x,ldx,nobs,s,lds,
               ncons,m,work(1),work(dim+1),dim,thetam)
   if (isigma == 0) {
      if (j <= nobs)
         jl=nobs+1
      else
         jl=ntotal+1
      for (i=j; i<=nobs; i=i+1) {
         esigma(i,j)=esigma(i,j)/(sigma(i)*sigma(j))
      }
      for (i=jl; i<=ntotal; i=i+1){
         esigma(i,j)=esigma(i,j)/sigma(j)
      }
   }
}
#
#                       Copy lower diagonal elements into upper diag.
#
do j=2,ntotal
   do i=1,j-1
      esigma(i,j)=esigma(j,i)
return
end
```

```
*****************************************************************
*                          DSGETA                               *
*****************************************************************

subroutine dsgeta(a,nobs,ntlmm,ntl,bigm,tsigma,qrtsla,workv,workm)
double precision a(ntlmm,ntlmm), tsigma(ntl,bigm),qrtsla(bigm),
                 workv(ntl),workm(ntl,ntl)
integer nobs,ntlmm,ntl,bigm
#
# Compute matrix a = Q ' * V * Q
#                    2        2
#
#                       Form V put in workm
#
do j=1,ntl {
   do i=1,ntl {
      if (i==j & i<=nobs)
         workm(i,j)=1.0d0
      else
         workm(i,j)=0.0d0
   }
}
call dsqtaq(tsigma,ntl,ntl,bigm,qrtsla,workm,ntl,a,ntlmm,
            workv,workm)
return
end


*****************************************************************
*                          DSGETB                               *
*****************************************************************

subroutine dsgetb(b,ntotal,nobs,ntl,ntlmm,bigm,
                  esigma,tsigma,qrtsla,workv,workm,
                  istant,nactiv)
double precision b(ntlmm,ntlmm),esigma(ntotal,ntotal),
                 tsigma(ntl,bigm),qrtsla(bigm),workv(1),
                 workm(ntl,ntl)
integer ntotal,nobs,ntl,ntlmm,bigm,istant(1),nactiv
#
# Compute
#
#       b = Q ' * E(1) * Q
#           2            2
#
#
#                     Copy rows and cols. of esigma that
#                     correspond to active constraints in
#                     workm, column by column
#
do j=1,nobs {
```

```
      call dcopy (nobs,esigma(1,j),1,workm(1,j),1)
      il=nobs
      do i=nobs+1,ntotal {
         if (istant(i-nobs) != 0) {
            il=il+1
            workm(il,j)=esigma(i,j)
         }
      }
   }
   jl=nobs
   do j=nobs+1,ntotal {
      if (istant(j-nobs) != 0) {
         jl=jl+1
         call dcopy(nobs,esigma(1,j),1,workm(1,jl),1)
         il=nobs
         do i=nobs+1,ntotal {
            if(istant(i-nobs) != 0) {
               il=il+1
               workm(il,jl)=esigma(i,j)
            }
         }
      }
   }
   #
   #                        Compute b
   #
   call dsqtaq(tsigma,ntl,ntl,bigm,qrtsla,workm,ntl,b,ntlmm,
               workv,workm)
   return
   end


   ************************************************************************
   *                                DSGRLA                                *
   ************************************************************************

   subroutine dsgrla(lambda,lamvec,nvalle,nvalri,nvalam,
                     bigeig,ntotal,zero)
   double precision lambda,lamvec(1),bigeig,zero
   integer nvalle,nvalri,nvalam,ntotal
   #
   # Construct regular grid around lambda (from unconstrained
   # problem)
   #
   if (lambda <= zero) {
      if (lambda <= -0.5d0) {
         nvalam=nvalle+1
         lamvec(nvalam)=dlog10(ntotal*bigeig*1.d3)
         lamvec(nvalle)=lamvec(nvalam)-2.d0
         lamvec(nvalle-1)=lamvec(nvalle)-1.d0
         lamvec(nvalle-2)=lamvec(nvalle)-2.d0
```

```
      for (i=nvalle-3; i>=1; i=i-1)
         lamvec(i)=lamvec(i+1)-0.1

   }
   else {
      nvalam=nvalri+1
      lamvec(1)=dlog10(zero)
      for (i=2; i<=nvalam; i=i+1)
         lamvec(i)=lamvec(i-1)+0.1
   }
}
else {
   nvalam=nvalle+nvalri+1
   il=nvalle+1
   lamvec(il)=dlog10(lambda)
   for (i=il+1; i<=nvalam; i=i+1)
      lamvec(i)=lamvec(i-1)+0.1
   for (i=il-1; i>=1; i=i-1)
      lamvec(i)=lamvec(i+1)-0.1
}
return
end


************************************************************************
*                              DSHE11                                  *
************************************************************************

subroutine dshell(hess1,ntotal,tsigqr,ldt,esigma,bigm,nobs,
                  qrtsia,workv,workm)
double precision hess1(ntotal,ntotal),tsigqr(ntotal,bigm),
                 esigma(ntotal,ntotal),qrtsia(bigm),workv(ntotal),
                 workm(ntotal,ntotal)
integer ntotal,ldt,bigm
#
# Form upper left hand corner of hess1
#
#                        Form:
#                        |E  E     E  E  |   |E  |
#                        | 11 11    11 12|   | 11|
#              workm =   |                | = |   | * |E    E  |
#                        |E  E     E  E  |   |E  |   | 11   12|
#                        | 21 11    21 21|   | 21|
#
do i=1,ntotal {
   do j=i,ntotal {
      workm(i,j)=ddot(nobs,esigma(i,1),ntotal,esigma(1,j),1)
      workm(j,i)=workm(i,j)
   }
}
#
```

```
#                       Obtain:
#
#                       Q ' * workm * Q
#                        2            2
#
call dsqtaq(tsigqr,ntotal,ntotal,bigm,qrtsia,workm,ntotal,
           hessl,ntotal,workv,workm)
return
end


************************************************************************
*                              DSHE12                                  *
************************************************************************

subroutine dshel2(hessl,ntotal,tsigqr,esigma,bigm,nobs,nmbigm,
                  qrtsia,tsigma,workm)
double precision hessl(ntotal,ntotal),tsigqr(ntotal,bigm),
                 esigma(ntotal,ntotal),qrtsia(bigm),
                 tsigma(ntotal,bigm),workm(ntotal,bigm)
integer ntotal,bigm,nobs,nmbigm
#
# Compute upper right hand corner of hessl  =
#
#                    |E  |
#               Q '* | 11| * T
#                2   |E  |    1
#                    | 21|
#
#                       Form
#
#                             |E  |
#                             | 11| * T
#                             |E  |    1
#                             | 21|
#
#                       put in workm
#
do j=1,bigm {
   do i=1,ntotal {
      workm(i,j)=ddot(nobs,esigma(1,i),1,tsigma(1,j),1)
   }
}
#
#                       Multiply by
#
#                            |Q '|
#                            | 1 |
#                            |Q '|
#                            | 2 |
#
```

```
job = 01000
do j=1,bigm
   call dqrsl(tsigqr,ntotal,ntotal,bigm,qrtsia,workm(1,j),
              dummy,workm(1,j),dummy,dummy,dummy,job,info)
#
#                     Put last nmbigm rows of workm which
#                     contain hesl2 into upper right hand
#                     and lower left hand corner of hessl
#
j2=bigm+1
do j=1,bigm {
   jl=nmbigm+j
   call dcopy(nmbigm,workm(j2,j),1,hessl(1,jl),1)
   call dcopy(nmbigm,workm(j2,j),1,hessl(jl,1),ntotal)
}
return
end


**************************************************************************
*                              DSHE22                                    *
**************************************************************************

subroutine dshe22(hessl,ntotal,nobs,bigm,nmbigm,tsigma)
double precision hessl(ntotal,ntotal),tsigma(ntotal,bigm)
integer ntotal,bigm,nmbigm,nobs
#
# Compute lower right hand corner of hessl =
#
#       T ' * T
#        1     1
#
do j=nmbigm+1,ntotal {
   do i=j,ntotal {
      hessl(i,j)=ddot(nobs,tsigma(1,i-nmbigm),1,tsigma(1,j-nmbigm),1)
      hessl(j,i)=hessl(i,j)
   }
}
return
end


**************************************************************************
*                              DSHEMU                                    *
**************************************************************************

subroutine dshemu(n,jthcol,hessl,hess2,ntotal,nmbigm,lambda,y,hy)
double precision y(1),hy(1),hessl(ntotal,1),hess2(nmbigm,1),lambda,
                 lamnob
integer n,jthcol,ntotal,nmbigm
common /block/nobs
```

```
#                       Note hess1(ntotal,ntotal),
#                            hess2(nmbigm,nmgigm)
#
# Multiply hessian times vector y or recover jth column of hessian
#
#
#       n       number of elements in y
#
#       jthcol  If jthcol = 0, hy will contain the product
#                              hessian*y
#               If jthcol > 0, hy will contain the jth column
#                              of the hessian
#
#       hess1   Double (ntotal,ntotal).
#               First part of hessian for spline problem.
#       hess2   Double (nmbigm,nmbigm).
#               Second part of hessian which gets multiplied
#               by lambda.
#
#       ntotal  Integer variable.
#               Row dimension of hess1.
#
#       nmbigm  Integer variable.
#               Row dimension of hess2.
#
#       lambda  Double Precision variable.
#               Current value of lambda.
#
#       y       Input vector required to compute hessian*y
#               when jthcol = 0.
#
#       hy      Output vector containing jth-col of hessian or
#               hessian*y depending on value of jthcol
#
lamnob=nobs*lambda
if (jthcol == 0) {
#                           compute hessian*y put in hy
   do j=1,nmbigm {
      hy(j)=ddot(ntotal,hess1(1,j),1,y,1)+
            lamnob*ddot(nmbigm,hess2(1,j),1,y,1)
   }
   do j=nmbigm+1,ntotal {
      hy(j)=ddot(ntotal,hess1(1,j),1,y,1)
   }
}
else {
   call dcopy(ntotal,hess1(1,jthcol),1,hy,1)
   if (jthcol <= nmbigm)
      call daxpy(nmbigm,lamnob,hess2(1,jthcol),1,hy,1)
}
return
```

```
end


*****************************************************************
*                            DSHES1                             *
*****************************************************************

subroutine dshes1(hess1,ntotal,nmbigm,nobs,tsigqr,qrtsia,tsigma,
                  bigm,esigma,work)
double precision hess1(ntotal,ntotal),tsigqr(ntotal,bigm),
                 qrtsia(bigm),tsigma(ntotal,bigm),
                 esigma(ntotal,ntotal),work(1)
integer ntotal,bigm
#
# Compute hess1
#
#                       Form first nmbigm X nmbigm of hess1
#
nw1=ntotal+1
call dshe11(hess1,ntotal,tsigqr,ntotal,esigma,bigm,nobs,qrtsia,
            work(1),work(nw1))
#
#                       Form upper right hand corner of hess1
#
call dshe12(hess1,ntotal,tsigqr,esigma,bigm,nobs,nmbigm,qrtsia,
            tsigma,work)
#
#               Form lower right hand corner of hess1=T '*T
#                                                     1   1
#
call dshe22(hess1,ntotal,nobs,bigm,nmbigm,tsigma)
return
end


*****************************************************************
*                            DSHOFP                             *
*****************************************************************

subroutine dshofp(p1,x,ldx,dim,nobs,s,lds,ncons,ntotal,m,
                  thetam,bigm,powers,ldp,coef,hofp)
double precision p1(6),x(ldx,1),s(lds,1),thetam,hofp,t1,t2,t3,
                 p2(6),coef(1),eij,prod
integer ldx,lds,ncons,ntotal,bigm,ldp,powers(ldp,bigm),dim
#
# This routine evaluates the spline h at the point p1.
#
# h(p1)=thetam * (t1 + t2) + t3
#
# where:
#           nobs
```

```
#       t1= SUM  coef(i)*e(x ,p1)
#           i=1             i
#
#           ncons
#       t2 = SUM  coef(nobs+i)*e(s ,p1)
#            i=1                 i
#
#            bigm
#       t3 = SUM  coef(ntotal+i)*phi (p1)
#            i=1                   i
#
# and
#       e(p2,p1) = t**(2*m-dim)        if dim is odd
#                = t**(2*m-dim)*ln(t)  if dim is even
#
#              t = || p2 - p1 ||.
#
#
#                       Compute t1
#
t1 = 0.d0
do j=1,nobs {
   do i=1,dim
      p2(i)=x(i,j)
   call dseij(p1,p2,dim,m,eij)
   t1=t1+coef(j)*eij
}
#
#                       Compute t2
#
t2 = 0.d0
do j=1,ncons {
   do i=1,dim
      p2(i)=s(i,j)
   call dseij(p1,p2,dim,m,eij)
   t2=t2+coef(nobs+j)*eij
}
#
#                       Compute t3
#
t3=0.d0
do j=1,bigm {
   prod=1.0d0
   do i=1,dim {
      ip=powers(i,j)
      prod=prod*(p1(i)**ip)
   }
   t3=t3+coef(ntotal+j)*prod
}
hofp=thetam * (t1 + t2) + t3
return
```

```
end


************************************************************************
*                              DSLIMA                                  *
************************************************************************

subroutine dslima(qplima,nobs,ntotal,bigm,nmbigm,tsigma,
                  tsigqr,qrtsia,esigma,work)
double precision qplima(nobs,ntotal),tsigma(ntotal,bigm),
                 tsigqr(ntotal,bigm),qrtsia(bigm),
                 esigma(ntotal,ntotal),work(1)
integer nobs,ntotal,bigm,nmbigm
#
#
# Compute matrix qplima which defines linear term
#
#                |    |E   | |
#                |Q '*| 11 | |
#                | 2  |E   | |
#     qplima' =  |    | 21 | |
#                |           |
#                |   T '     |
#                |    1      |
#
#
job=01000
i1=nmbigm+1
i2=bigm+1
do i=1,nobs {
   call dqrsl(tsigqr,ntotal,ntotal,bigm,qrtsia,
              esigma(1,i),dmy,work(1),dmy,dmy,dmy,
              job,info)
   call dcopy(nmbigm,work(i2),1,qplima(i,1),nobs)
   call dcopy(bigm,tsigma(i,1),ntotal,qplima(i,i1),nobs)
}
return
end


************************************************************************
*                              DSLIVE                                  *
************************************************************************

subroutine dslive(qplive,ntotal,z,nobs,qplima)
double precision qplive(ntotal),z(nobs),qplima(nobs,ntotal)
integer ntotal,nobs
#
# Compute -z'*qplima = linear term in quadratic function
#
do i=1,ntotal
```

```
   qplive(i)=-ddot(nobs,z,1,qplima(1,i),1)
return
end


**********************************************************************
*                              DSLMIN                                *
********************************************************************** ****

subroutine dslmin(voflam,nvalam,lminin)
double precision voflam(nvalam)
integer lminin(nvalam)
#
# This routine determines for which indices
# in voflam there is a local minimum and sets
# the corresponding entry of lminin to 1
#
if (nvalam <= 3) return
if (voflam(1) < voflam(2) & (voflam(1) > 0))
   lminin(1) = 1
else
   lminin(1) = 0
if ((voflam(nvalam)<voflam(nvalam-1)) & (voflam(nvalam) > 0))
   lminin(nvalam) = 1
else
   lminin(nvalam) = 0
do i=2,nvalam-1 {
   if ((voflam(i)<=voflam(i-1))&(voflam(i)<=voflam(i+1))&(voflam(i)>0))
      lminin(i) = 1
   else
      lminin(i) = 0
}
return
end


**********************************************************************
*                              DSMACH                                *
**********************************************************************

subroutine dsmach( wmach )
double precision   wmach(15)
#
#  mchpar  must define the relevant machine parameters as follows.
#     wmach(1)  = nbase  = base of floating-point arithmetic.
#     wmach(2)  = ndigit = no. of base wmach(1) digits of precision.
#     wmach(3)  = epsmch = floating-point precision.
#     wmach(4)  = rteps  = sqrt(epsmch).
#     wmach(5)  = flmin  = smallest positive floating-point number.
#     wmach(6)  = rtmin  = sqrt(flmin).
#     wmach(7)  = flmax  = largest positive floating-point number.
```

```
#      wmach(8)  = rtmax  = sqrt(flmax).
#      wmach(9)  = undflw = 0.0 if underflow is not fatal,+ve otherwise.
#      wmach(10) = nin    = standard file number of the input stream.
#      wmach(11) = nout   = standard file number of the output stream.
#
double precision   dsqrt
#
nbase     = 2
ndigit    = 56
wmach(1)  = nbase
wmach(2)  = ndigit
wmach(3)  = wmach(1)**(1 - ndigit)
wmach(4)  = dsqrt(wmach(3))
wmach(5)  = wmach(1)**(-128)
wmach(6)  = dsqrt(wmach(5))
wmach(7)  = wmach(1)**126
wmach(8)  = dsqrt(wmach(7))
wmach(9)  = 0.0
nin       = 5
nout      = 8
wmach(10) = nin
wmach(11) = nout
return
end


*****************************************************************************
*                                  DSPOLY                                   *
*****************************************************************************

subroutine dspoly (dim,m,powers,ldp)
#
# Record powers of polynomials.
#
integer powers(ldp,1),p1,p2,p3,p4,p5,p6,dim,m,ldp,count
#
count=0
if (dim == 1) {
   do p1=0,m-1 {
      count=count+1
      powers(1,count)=p1
   }
   return
}
if (dim == 2) {
   do p1=0,m-1
      do p2=0,m-1-p1 {
         count=count+1
         powers(1,count)=p1
         powers(2,count)=p2
      }
```

```
   return
}
if (dim == 3) {
   do p1=0,m-1
      do p2=0,m-1-p1
         do p3=0,m-1-p1-p2 {
            count=count+1
            powers(1,count)=p1
            powers(2,count)=p2
            powers(3,count)=p3
         }
   return
}
if (dim == 4) {
   do p1=0,m-1
      do p2=0,m-1-p1
         do p3=0,m-1-p1-p2
            do p4=0,m-1-p1-p2-p3 {
               count=count+1
               powers(1,count)=p1
               powers(2,count)=p2
               powers(3,count)=p3
               powers(4,count)=p4
            }
   return
}
if (dim == 5) {
   do p1=0,m-1
      do p2=0,m-1-p1
         do p3=0,m-1-p1-p2
            do p4=0,m-1-p1-p2-p3
               do p5=0,m-1-p1-p2-p3-p4 {
                  count=count+1
                  powers(1,count)=p1
                  powers(2,count)=p2
                  powers(3,count)=p3
                  powers(4,count)=p4
                  powers(5,count)=p5
               }
   return
}
if (dim == 6) {
   do p1=0,m-1
      do p2=0,m-1-p1
         do p3=0,m-1-p1-p2
            do p4=0,m-1-p1-p2-p3
               do p5=0,m-1-p1-p2-p3-p4
                  do p6=0,m-1-p1-p2-p3-p4-p5 {
                     count=count+1
                     powers(1,count)=p1
                     powers(2,count)=p2
```

```
                        powers(3,count)=p3
                        powers(4,count)=p4
                        powers(5,count)=p5
                        powers(6,count)=p6
                     }
   return
}
end


*************************************************************************
*                              DSQTAQ                                   *
*************************************************************************

subroutine dsqtaq(t,ldt,n,m,qraux,a,lda,q2taq2,ldqaq,workv,workm)
double precision t(ldt,m),qraux(m),a(lda,n),workv(n),
                 q2taq2(ldqaq,n-m),workm(lda,n)
integer ldt,n,m,lda,ldqaq
#
# This routine forms the product
#
#       |Q '|
#       | 1 | * A |Q   Q | = Q' * A * Q
#       |Q '|     | 1   2|
#       | 2 |
#
# and stores Q ' * A * Q
#             2       2
#
# in the matrix q2taq2.
#
# To save storage the matrix a can also be used as workm
# in which case on return a is destroyed
#
#
#                       Get
#                          workm=Q'*A
#
job=01000
do j=1,n
   call dqrsl(t,ldt,n,m,qraux,a(1,j),dmy,workm(1,j),dmy,dmy,
              dmy,job,info)
#
#                       Post-multiply the last n-m rows by Q
#                                                          2
#                       and put in Q'AQ, this is done by
#                       premultiplying the last n-m columns of
#                       workm' by Q' and then taking the last
#                       n-m rows of the resulting matrix
#
nminm=n-m
```

```
for (i=m+1; i<=n; i=i+1) {
   il=i+m
#                        copy ith. row of workm to
#                        workv(= ith.col. of workm')
   call dcopy(n,workm(i,1),lda,workv,1)
#                        put Q'*ith row of workm in workv
   call dqrsl(t,ldt,n,m,qraux,workv,dmy,workv,dmy,dmy,dmy,job,info)
#                        copy last n-m els. of workv to (i-m)th col.
#                        of q2taq2
   nminm=n-m+1
   call dcopy(nminm,workv(m+1),1,q2taq2(1,i-m),1)
}
return
end


*************************************************************************
*                              DSRD1                                    *
*************************************************************************

subroutine dsrd1(ntotal,nobs,ncons,bigm,dim,nmbigm,hess1,
                 hess2,qplima,qpcons,tsigqr,qrtsia,
                 esigma,ierror)
double precision hess1(ntotal,ntotal),hess2(nmbigm,nmbigm),
                 qplima(nobs,ntotal),qpcons(ncons,ntotal),
                 tsigqr(ntotal,bigm),qrtsia(bigm),
                 esigma(ntotal,ntotal)
integer ntotal,nobs,ncons,bigm,dim,ierror
#
# Read intermediate results from file associated to unit 1
#
# If first input record does not agree with the values from
# the calling program ierror is set to 10 and nothing is done
#
# If there is insufficient data in file ierr is set to 11
# and the routine returns at that point
#
read (1,end=100) nt,no,nc,mbig,nd
if (no != nobs | nc != ncons | mbig != bigm | nd != dim){
   ierror = 10
   return
}
nmbigm=ntotal-bigm
read (1,end=100) ((hess1(i,j),i=1,ntotal),j=1,ntotal)
read (1,end=100) ((hess2(i,j),i=1,nmbigm),j=1,nmbigm)
read (1,end=100) ((qplima(i,j),i=1,nobs),j=1,ntotal)
read (1,end=100) ((qpcons(i,j),i=1,ncons),j=1,ntotal)
read (1,end=100) ((tsigqr(i,j),i=1,ntotal),j=1,bigm)
read (1,end=100) (qrtsia(i),i=1,bigm)
read (1,end=100) ((esigma(i,j),i=1,ntotal),j=1,ntotal)
ierror = 0
```

```
return
#
#                        error return - insufficient data
#
100 ierror = 11
return
end


*****************************************************************************
*                                  DSRDS                                    *
*****************************************************************************

subroutine dsrds(x,ldx,dim,nobs,s,lds,ncons,z,m,nvalam,istate,
                 nactiv,iactch,powers,ldp,bigm,coef,hhat,
                 voflam,lamvec)
double precision x(ldx,1),s(lds,1),z(1),coef(1),hhat(1),
                 voflam(1),lamvec(1)
integer ldx,dim,nobs,lds,ncons,m,nvalam,istate(1),nactiv,iactch(1),
       powers(ldp,1),ldp,bigm
#
# Read solution obtained by dscomp in file splsol
#
open (3,file='splsol',form='unformatted')
rewind (3)
read (3) dim,nobs,ncons,m,bigm,nvalam,nactiv,thetam
read (3) ((x(i,j),i=1,dim),j=1,nobs)
read (3) ((s(i,j),i=1,dim),j=1,ncons)
read (3) (z(i),i=1,nobs)
read (3) (istate(i),i=nobs+ncons+1,nobs+2*ncons)
read (3) (iactch(i),i=1,nvalam)
read (3) ((powers(i,j),i=1,dim),j=1,bigm)
read (3) (coef(i),i=1,nobs+ncons+bigm)
read (3) (hhat(i),i=1,nobs)
read (3) (voflam(i),i=1,nvalam)
read (3) (lamvec(i),i=1,nvalam)
return
end


*****************************************************************************
*                                  DSSOLV                                   *
*****************************************************************************

subroutine dssolv(hess1,ntotal,hess2,nmbigm,bigm,qplima,
                  qplive,qpcons,ncons,nobs,tsigma,
                  qrtsil,lamvec,voflam,indlam,coef,ioptv,iqpout,
                  nvalle,nvalri,nvalam,zero,finity,bl,bu,
                  start,istate,istant,iwork,iactch,
                  wmach,bigeig,itmax,hhat,z,esigma,nactiv,eigen,
                  work,ierr)
```

```
external dshemu
common /block/nobser
logical start
integer ntotal,nmbigm,bigm,ncons,nobs,ntotal,indlam,ioptv,iqpout,
        nvalle,nvalri,nvalam,istate(1),istant(1),iwork(1),
        iactch(1),itmax,nactiv
double precision hess1(ntotal,1),hess2(nmbigm,1),qplima(nobs,ntotal),
                 qplima(nobs,ntotal),qplive(ntotal),
                 tsigma(ntotal,bigm),qrtsil(bigm),lamvec(nvalam),
                 voflam(nvalam),coef(ntotal),zero,finity,
                 bl(ncons+ntotal),bu(ncons+ntotal),wmach(15),
                 bigeig,hhat(nobs),z(nobs),esigma(ntotal,ntotal),
                 work(1),vnum,lambda,eigen(1)
#
# This routine solves the actual problem optimizing with respect to
# lambda if necessary.  The routine qpfc is called from this routine
#
ierr=0
lambda=lamvec(1)
if (ioptv == 0) {
   nvalle=0
   nvalri=0
   nvalam=1
   lamvec(1)=dlog10(lambda)
}
else
#                          Construct regular grid of values of log(lambda)
   call dsgrla(lambda,lamvec,nvalle,nvalri,nvalam,bigeig,ntotal,zero)
msglvl=iqpout
msglvq=iqpout
maxact=ncons
n=ntotal
nclin=ncons
nctotl=n+nclin
liwork=nctotl
lwork=n*n+8*n+nclin+(nclin+1)*(nclin+1)+n*(nclin+1)+1
nfree = n
featol=wmach(4)
#
#                          set coef to initial guess
#                          (Recall that work(1) is used as coef
#                          to have in coef a minimum at every stage
#
do i=1,n
   work(i)=0.1d0
#
#                          initialize istant
#
do i=1,nclin
   istant(i)=-1
#
```

```
#                       initialize first n rows of bl,bu and istate
#
do i=1,n {
   bl(i)=-finity
   bu(i)= finity
   istate(i)=0
}
#
#                       for each value of lambda in lamvec solve
#                       quadratic programming problem and evaluate
#                       generalized cross-validation
#
iactch(nvalam)=0
vlant=finity
nobser=nobs
for (ival=nvalam; ival>=1; ival=ival-1) {
   nw0=ntotal+1
   nw1=ncons+nw0
   nw2=ntotal+ncons+nw1
   lambda=10.d0**lamvec(ival)
   call qpfc(dshemu,hess1,hess2,ntotal,nmbigm,lambda,start,iter,itmax,
             liwork,lwork,msglvl,msglvq,maxact,n,nclin,nctotl,nclin,
             inform,nactiv,nfree,istate,iwork,finity,finity,featol,
             qpcons,work(nw0),bl,bu,work(nw1),qplive,work,work(nw2),
             wmach)
   if (inform > 0) {
      voflam(ival)=-1.0d0
      if (inform == 200) inform=5
      iactch(ival)=inform+1
      next
   }
   if (nvalam == 1) {
      call dcopy(n,work,1,coef,1)
      indlam=1
      break
   }
   nt1=nobs+nactiv
   ntlmm=nt1-bigm
   nw1=nw0+ntlmm*ntlmm
   nw2=nw1+ntlmm*ntlmm
   vnum=0.d0
#
#                       Compute sumn of squares of the
#                       residuals to form numerator of
#                       Cross-validation function
#
   do i=1,nobs {
#
#                       compute hhat(z(i) put in work(nw0+i-1)
#
      work(nw0+i-1)=ddot(ntotal,qplima(i,1),nobs,work,1)
```

```
      temp=work(nw0+i-1)-z(i)
      vnum=vnum+temp*temp
   }
#
#                        Compute generalized cross-validation
#
   call dsvofl(voflam(ival),lambda,vnum,ntotal,bigm,nobs,ncons,nmbigm,
               qplima,qpcons,esigma,tsigma,qrtsil,istate,istant,
               nactiv,ntl,ntlmm,idifer,neig,eigen,work(nw0),work(nw1),
               work(nw2),ierr,iwork)
   if (ierr != 0) return
#
#                        If set of active constraints changed, change
#                        the value of iactch(ival)
#
   if (idifer == 1 & ival < nvalam) {
      if (iactch(ival+1) == 0)
         iactch(ival)=1
      else
         iactch(ival)=0
   }
#
#                        Set start to .true. to use current set of
#                        active constraints as guess for next value
#                        of lambda
#
   start=.true.
#
#                        If current value of v(lambda) is smaller
#                        previous one, store current value of ival
#                        in indlam and current solution in coef
#
   if (voflam(ival) < vlant) {
      vlant=voflam(ival)
      call dcopy(n,work,1,coef,1)
      indlam=ival
   }
#
#                        Go to solve q.p. problem for next value
#                        of lambda
}
#                        Compute hhat
#
do i=1,nobs
   hhat(i)=ddot(ntotal,qplima(i,1),nobs,coef,1)
return
end
```

```
*****************************************************************************
*                                  DSTETA                                   *
*****************************************************************************

subroutine dsteta (dim,m,thetam)
#
#     get theta sub m for dim,m
#
#
#
integer dim,m
double precision thetam
integer rtab(36),p
double precision pi
data pi / 3.14159265358979323846d0 /
data rtab /  12, -240, 10080, -725760, 79833600, -12454041600,
             8, -128, 4608, -294912, 29491200, -4246732800,
             -8, 96, -2880, 161280, -14515200, 1916006400,
             1, 64, -1536, 73728, 1, 1,
             1, -64, 1152, 1, 1, 1,
             1, 1, 768, 1, 1, 1 /
#
l = 6*dim + m - 7
p = dim/2
thetam = 1.0d0 / (dfloat(rtab(l)) * (pi**p))
return
end
```

```
*****************************************************************
*                            DSTSIG                             *
*****************************************************************

subroutine dstsig(x,ldx,s,lds,powers,ldp,tsigma,ntotal,bigm,
                  nobs,ncons,dim,isigma,sigma)
double precision x(ldx,nobs),s(lds,ncons),
                 sigma(nobs), tsigma(ntotal,bigm)
integer ldx,lds,dim,nobs,isigma, powers(ldp,bigm),ldp,bigm,
        ntotal,ncons
#
# Compute:
#             |SIGMA   0| |T |
# tsigma  =   |         | | 1|
#             |0       I| |T |
#             |         | | 2|
#
#
do l=1,bigm {
   do j=1,nobs {
      prod=1.0d0
      do i=1,dim {
         ip=powers(i,l)
         if (ip == 0) next
         prod = prod*(x(i,j))**ip
      }
#                          divide by sigma if needed
      if (isigma == 0) prod=prod/sigma(j)
      tsigma(j,l)=prod
   }
   do j=1,ncons {
      prod = 1.0d0
      do i=1,dim {
         ip=powers(i,l)
         if (ip == 0)
           next
         else
            prod=prod*(s(i,j))**ip
      }
      tsigma (nobs+j,l)=prod
   }
}
return
end
```

```
*****************************************************************
*                          DSVOFL                               *
*****************************************************************

subroutine dsvofl(voflam,lambda,vnum,ntotal,bigm,
                  nobs,ncons,nmbigm,qplima,qpcons,esigma,
                  tsigma,qrtsil,istate,istant,nactiv,ntl,ntlmm,
                  idifer,neig,eigen,b,a,work,ierr,iwork)
double precision voflam,vnum,qplima(nobs,ntotal),
                 qpcons(ncons,ntotal),esigma(ntotal,ntotal),
                 tsigma(ntl,bigm),qrtsil(bigm),eigen(1),
                 b(ntlmm,ntlmm),a(ntlmm,ntlmm),denom,
                 lambda,lam,work(1),eigi
integer ntl,ntlmm,ntotal,nobs,ncons,nmbigm,nactiv,neig,iwork(1),
        istate(1),istant(1),bigm
#
# Compute generalized cross-validation function for
# constrained problems
#
lam=lambda
idifer=0
#
#                       Check if active constraints changed
#                       and update istant if necessary
#
do i=1,ncons {
   if (istate(ntotal+i) != istant(i)) {
      istant(i)=istate(ntotal+i)
      idifer=1
   }
}
#
#                       If constraints did not change skip
#                       generalized eigenvalue problem
#
if (idifer == 0) go to 100
#
#                       Recover T(1) from qplima and qpcons
#                       and put in tsigma
#
do j=1,bigm {
   call dcopy(nobs,qplima(1,nmbigm+j),1,tsigma(1,j),1)
}
il=nobs
do i=1,ncons {
   if (istant(i) != 0) {
      il=il+1
      call dcopy(bigm,qpcons(i,nmbigm+1),ncons,tsigma(il,1),ntl)
   }
}
#                       Obtain Q-R decomposition of tsigma
```

```
#                       which contains T(1)
#
job = 0
call dqrdc(tsigma,ntl,ntl,bigm,qrtsil,jpvt,work,job)
#
#                       Compute
#
#                       b = Q ' * E(1) * Q
#                            2           2
#
call dsgetb(b,ntotal,nobs,ntl,ntlmm,bigm,esigma,tsigma,
           qrtsil,work(1),work(ntl+1),istant,nactiv)
#
#                       Get
#
#                       a = Q ' * V * Q
#                            2       2
#
call dsgeta(a,nobs,ntlmm,ntl,bigm,tsigma,qrtsil,work,
           work(ntl+1))
#
#                       Solve generalized eigenvalue problem
#
call dseige(ntlmm,a,b,eigen,work,ierr)
if (ierr == 7*ntlmm+1) {
#                       b is not pos. def. ierr=20
   ierr=20
   return
}
#
#
#                       Note: eigen has the eigenvalues
#                       of       -1
#                              B  * A
#                       in ascending order
#
100 continue
#
#                       compute denominator of voflam:
#
#                       denom = (1/nobs)trace(I-A(lambda))
#                                        ntlmm
#                       =(lambda*(SUM
#                                        i=1
#                                                          2
#                           ((eigen(i)/(1+nobs*lambda*eigen(i)))
#
#
#
denom=0.d0
#                       First compute trace(I-A(lambda))
```

```
#
neig=0
do i=1,ntlmm {
   eigi=eigen(i)
   if (eigi > 0.0d0) {
      neig=neig+1
      denom=denom+eigi/(1.0d0+dfloat(nobs)*lam*eigi)
   }
}
#
#                        neig=number of positive eigenvalues
#
denom = ((denom*lam)/dfloat(nobs))*denom*lam
voflam=vnum/denom
return
end


**********************************************************************
*                              DSWRT1                                *
**********************************************************************

subroutine dswrt1(ntotal,nobs,ncons,bigm,dim,nmbigm,hess1,
                  hess2,qplima,qpcons,tsigqr,qrtsia,
                  esigma)
double precision hess1(ntotal,ntotal),hess2(nmbigm,nmbigm),
                  qplima(nobs,ntotal),qpcons(ncons,ntotal),
                  tsigqr(ntotal,bigm),qrtsia(bigm),
                  esigma(ntotal,ntotal)
integer ntotal,nobs,ncons,bigm,dim,nmbigm
#
# Write intermediate results from file associated to unit 1
#
#
write (1) ntotal,nobs,ncons,bigm,dim
nmbigm=ntotal-bigm
write (1) ((hess1(i,j),i=1,ntotal),j=1,ntotal)
write (1) ((hess2(i,j),i=1,nmbigm),j=1,nmbigm)
write (1) ((qplima(i,j),i=1,nobs),j=1,ntotal)
write (1) ((qpcons(i,j),i=1,ncons),j=1,ntotal)
write (1) ((tsigqr(i,j),i=1,ntotal),j=1,bigm)
write (1) (qrtsia(i),i=1,bigm)
write (1) ((esigma(i,j),i=1,ntotal),j=1,ntotal)
return
end
```

```
*****************************************************************
*                          DSWRTS                               *
*****************************************************************

subroutine dswrts(x,ldx,dim,nobs,s,lds,ncons,z,m,nvalam,istate,
                  nactiv,iactch,powers,ldp,bigm,coef,hhat,
                  voflam,lamvec)
double precision x(ldx,1),s(lds,1),z(1),coef(1),hhat(1),
                 voflam(1),lamvec(1)
integer ldx,dim,nobs,lds,ncons,m,nvalam,istate(1),nactiv,iactch(1),
        powers(ldp,1),ldp,bigm
#
# Write solution obtained by dscomp in file splsol
#
open (3,file='splsol',form='unformatted')
rewind (3)
write (3) dim,nobs,ncons,m,bigm,nvalam,nactiv,thetam
write (3) ((x(i,j),i=1,dim),j=1,nobs)
write (3) ((s(i,j),i=1,dim),j=1,ncons)
write (3) (z(i),i=1,nobs)
write (3) (istate(i),i=nobs+ncons+1,nobs+2*ncons)
write (3) (iactch(i),i=1,nvalam)
write (3) ((powers(i,j),i=1,dim),j=1,bigm)
write (3) (coef(i),i=1,nobs+ncons+bigm)
write (3) (hhat(i),i=1,nobs)
write (3) (voflam(i),i=1,nvalam)
write (3) (lamvec(i),i=1,nvalam)
return
end


*****************************************************************
*                          DSZDIS                               *
*****************************************************************

subroutine dszdis (zdata,nobs,sigma)
#
#***** scale zdata values *****
#***** dszdis divides zdata by sigma *****
#
implicit double precision (a-h,o-z)
dimension ydata(1),sigma(1)
#
#
#***** divide by sigma *****
do i=1,nobs
   ydata(i)=ydata(i)/sigma(i)
return
end
```

```
*****************************************************************************
*                                  DSZMUS                                   *
*****************************************************************************

subroutine dszmus (zdata,nobs,sigma)
#
#***** scale zdata values *****
#***** dszmus multiplies zdata by sigma *****
#
implicit double precision (a-h,o-z)
dimension zdata(1),sigma(1)
#
#
#
#***** multiply by sigma *****
do i=1,nobs
   zdata(i)=zdata(i)*sigma(i)
return
end
```

BIBLIOGRAPHY

Adams, R. A. (1975). *Sobolev Spaces*, Academic Press .

Aitchison, J., Habbema, J. D. F., and Kay, J. W. (1977). A critical comparison of two methods of statistical discirimination, *Applied Statistics 26*, pp. 15-25. .

Akhiezer, N. I. and Glazman, I. M. (1961). *Theory of Linear Operators in Hilbert Space, Vol. 1*, Frederick Ungar Publishing Co. .

Anderson, J. A. (1972). Separate sample logistic discrimination, *Biometrika 59*, 1, pp. 19-35.

Anderson, J. A. and Senthilselvan, A. (1980). Smooth estimates for the hazard function, *Journal of the Royal Statistical Society, Series B 42*, 3, pp. 322-327.

Anderson, J. A. and Blair, V. (1982). Penalized maximum likelihood estimation in logistic regression and discrimination, *Biometrika 69*, 1, pp. 123-136.

Aronszajn, N. (1950). Theory of reproducing kernels, *Transactions American Mathematical Society 58*, pp. 337-404.

Aubin, J. P. (1979). *Applied Functional Analysis*, John Wiley and Sons, New York.

BMDP, (1975). *Biomedical Computer Programs*, University of California Press, Berkeley and Los Angeles.

Bates, D. and Wahba, G.(1983). Computational methods for generalized cross-validation with large data sets. In *Treatment of Integral Equations by Numerical Methods*, C. T. H. Baker and G. F. Miller, eds. Academic Press, London , pp. 283-296.

Bazaraa, M. S. and Shetty, C. M. (1979). *Nonlinear Programming Theory and Algorithms*, John Wiley & Sons, New York.

Boyle, J. M., Dongarra, J. J., Garbow, B. S., and Moler, C. B.(1977). Matrix eigen-system routines - EISPACK guide extension. In *Lecture Notes in Computer Science*, G. Goos and J. Hartmanis eds. Springer-Verlag, New York ,

Chi, P. Y. and Van Ryzin, J.(1977). A simple histogram method for non-parametric classification . In *Classification and Clustering*, J. Van Ryzin, ed. , pp. 395-421.

Cox, D.(1982). Convergence rates for multivariate smoothing spline functions. Mathematics Research Center, UW-Madison, TR#2437 .

Cox, D. R.(1966). Some procedures connected with the logistic qualitative response curve. In *Research Papers in Statistics*, F. N. David, ed. John Wiley & Sons, New York , pp. 55-71.

Craven, P. and Wahba, G. (1979). Smoothing Noisy Data with Spline Functions. *Numerisch Mathematic 31*, pp. 377-403.

Daniel, J. W. (1971). *The Approximate Minimization of Functionals*, Prentice-Hall .

Day, N. E. and Kerridge, D. F. (1967). A general maximum likelihood discriminant., *Biometrics 23*, pp. 313-323.

Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W. (1979). *Linpack Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia.

Duchon, J. (1976). Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces, *R. A. I. R. O. Analyse Numerique 10*, 12, pp. 5-12.

Dyn, N. and Wahba, G. (1982). On the estimation of functions of several variables from agregated data, *SIAM Journal of Mathematical Analysis 13*, pp. 134-152.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems, *Annals of Eugenics 7*, pp. 179-188.

Fix, E. and Hodges, J. L.(1951). Discriminatory analysis, nonparametric discrimination: consistency properties. Report No. 4, Project No. 21-49-004, USAF School of Aviation Medicine, Brooks Air Force Base, Texas .

Gilbert, E. S. (1969). The effect of unequal variance-covariance matrices on Fisher's linear discriminant function, *Biometrics 25*, pp. 505-515.

Gill, P. E., Gould, N. I. M., Murray, W., Saunders, M. A., and Wright, M. H.(1982). Range-space methods for convex quadratic programming. T. R. SOL 82-14, Systems Optimization Laboratory, Department of Operations Research, Stanford University .

Glick, N. (1972). Sample-based classification procedures derived from density estimators. , *Journal of the American Statistical Association 67*, pp. 116-122.

Goldstein, M. (1975). Comparison of some density estimate classification procedures, *Journal of the American Statistical Association 70*, pp. 666-669.

Golub, G., Heath, M., and Wahba, G. (1979). Generalised cross validation as a method for choosing a good ridge parameter, *Technometrics 31*, pp. 315-224.

Good, I. J. and Gaskins, R. A. (1971). Non-parametric roughness penalties for Probality Densities, *Biometrika 58*, pp. 255-277.

Habbema, J. D. F., Hermans, J., and Van den Broek, K.(1974). A stepwise discriminant analysis program using density estimation. In *COMPSTAT 1974, Proceedings in Computational Statistics*, G. Bruckmann, ed. Physica Verlag, Wien. ,

Hermans, J. and Habbema, J. D. F. (1975). Comparison of five methods to estimate posterior probabilities, *EDV in Medizin and Biologie 6*, pp. 14-19.

Hermans, J. and Habbema, J. D. F. (1976). *Manual for the ALLOC Discriminant Analysis Programs*, Department of Medical Statistics University of Leiden, Netherlands.

IMSL. (1983). *IMSL Libraries, Edition 10 (To appear)*, International Mathematical and Statistical Libraries, Inc., Houston, Texas.

Kernighan, B. W. and Plauger, P. J. (1976). *Software Tools*, Addison-Wesley .

Lachenbruch, P. A. and Mickey, M. R. (1968). Estimation of error rate in discriminant analysis, *Technometrics 10*, pp. 1-11.

Lachenbruch, P. A. and Goldstein, M. (1979). Discriminant analysis, *Biometrics 35*, pp. 69-85.

Luenberger, D. G. (1969). *Optimization by Vector Space Methods*, John Wiley and Sons, Inc., New York.

Madison Academic Computing Center, (1981). *Multi-dimensional Spline Smoothing Routines*, University of Wisconsin, Madison, Wisconsin.

Marks, S. and Dunn, O. J. (1974). Discriminant functions when covariance matrices are unequal, *Journal of the American Statistical Association 69*, pp. 555-559.

Meinguet, J.(1978). Multivariate interpolation at arbitrary points made simple.. Report No. 118, Institute de Mathematique Pure et Appliquee, Universite Catholoque de Louvain (to appear in A. Agnew. Math. Phys.) .

Meinguet, J.(1979). An intrinsic approach to multivariate spline interpolation at arbitrary points.. In *Proceedings of the NATO Advanced Study Institute on Polynomial and Spline Approximation*, B. Sahney ed. Calgary ,

Ortega, J. M. and Rheinbold, W. C. (1970). *Iterative Solutions of Non-linear Equations in Several Variables*, Academic Press .

Penrose, L. S. (1945). Discrimination between normal and psychotic subjects by revised examination, *M. Bull. Canad. Psychol. Ass. 5*, pp. 37-40.

Reinsch, C. H. (1967). Smoothing by spline functions I, *Numerische Mathematik 10*, pp. 177-183.

Remme, J., Habbema, J. D. F. and Hermans, J. (1980). A simulative comparison of linear, quadratic and kernel discrimination, *Journal of Statistical Comp. and Simulation 11*, pp. 87-105.

Schoenberg, I. J. (1964). Spine functions and the problem of graduation, *Proceedings of National Academie of Sciences 52* 4, pp 947-950.

Silverman, B. W. (1978). Density ratios, empirical likelihood and cot death, *Applied Statistics 27*, pp. 26-33.

Silverman, B. W. (1982). On the estimation of a probility density function by the maximum penalized likelihood method, *The Annals of Statistics 10*, 3, pp. 795-810.

Smith, C.A.B. (1947). Some examples of discrimination, *Annals of Eugenics 13*, pp. 272-282.

Tapia, R. A. and Thompson, J.R. (1978). *Nonparametric Probability Density Estimation*, John Hoplins University Press, Baltimore.

Van Ness, J. W. and Simpson, C (1976). On the effects of dimension in discriminant analysis, *Technometrics 18*, pp. 175-187.

Van Ness, J. W. (1980). On the dominance of non-parametric Bayes rule discriminant algorithms in high dimensions, *Pattern Recognition 12*, pp. 355-368.

Wahba, G. and Wold, S. (1975) A completely Automatic French Curve: Fitting Spline Functions by Cross Validation, *Communications in Statistics 4*, pp. 1-17.

Wahba, G.(1977). Optimal smoothing of density estimates. In *Classification and Clustering*, J. Van Ryzin, ed. Academic Press, New York , pp. 423-458.

Wahba, G. (1978). Improper priors, spline smoothing and the problem of guarding against model errors in regression., *J. Roy. Stat. Soc. B. 40*, 3, pp. 364-372.

Wahba, G.(1979a). Convergence rates of thin plate smoothing splines. In *Smoothing Techniques for Curve Estimation. Lecture Notes in Mathematics No. 757* Th. Gasser and M. Rosenblatt, eds. Springer-Verlag, New York , pp.233-245.

Wahba, G. (1979b). How to smooth curves and surfaces with splines and cross-validation, *Proceeding of the 24th Design of Experiments Conference. U. S. Army Research Office, Rep. 79-2*, pp. 167-192.

Wahba, G. and Wendelberger, J (1980). Some new mathematical methods for varitional objective analysis using spline and cross-validation, *Monthly Weather Review 108*, 8 pp. 1122-1143.

Wahba, G.(1980). Ill posed problems: numerical and statistical methods for mildly, moderately and severely ill posed problems with noisy data. Statistics Dept., UW-Madison, TR#595, to appear in the *Proceedings of the International Conference on Ill Posed Problems*, M. Z. Nashed, ed. .

Wahba, G.(1981a). Cross validation and constrained regularization methods for mildly ill posed problems. Statistics Dept., UW-Madison, TR#629 .

Wahba, G. (1981b). Data-based optimal smoothing of orthogonal series density estimates, *Annals of Statistics 9*, pp. 146-156.

Wahba, G.(1982). Constrained regularization for ill-posed linear operator equations, with applications in meteorology and medicine. In *Third Purdue Symposium on Statistical Decision Theory, Vol. 2*, S. S. Gupta and J. O. Berger, eds. Academic Press, New York , pp. 383-418.

Wegman, E. J. and Wright, I. W. (1983). Splines in statistics, *Journal of the American Statistical Association 78*, 382, pp. 351-365.

Wendelberger, J.(1981). The computation of Laplacian smoothing splines with examples. Statistics Dept., UW-Madison, TR#648 .

Wendelberger, J.(1982). Smoothing Noisy Data with Multi-dimensional Splines and Generalized Cross-validation. Ph. D. Thesis, Department of Statistics, University of Wisconsin, Madison .

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report No. 725	2. GOVT ACCESSION NO AD-A134515	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ESTIMATION OF POSTERIOR PROBABILITIES USING MULTIVARIATE SMOOTHING SPLINES AND GENERALIZED CROSS-VALIDATION		5. TYPE OF REPORT & PERIOD COVERED Scientific Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Miguel A. Villalobos		8. CONTRACT OR GRANT NUMBER(s) ONR N00014-77-C-0675 ARO DAAG29-80-K-0042
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Statistics University of Wisconsin, 1210 W. Dayton St. Madison, WI 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, Arlington, VA Army Research Office, Research Triangle Park, N.C.		12. REPORT DATE September 1983 13. NUMBER OF PAGES 155
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

constrained smoothing splines; generalized cross validation; thin plate splines; posterior probabioities; classification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see attached

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

A nonparametric estimate for the posterior probabilities in the classification problem using multivariate smoothing splines is proposed. This estimate presents a nonparametric alternative to logistic discrimination and to survival curve estimation. It is useful in exploring properties of the data and in presenting them in a way comprehensible to the layman.

The estimate is obtained as the solution to a constrained minimization problem in a reproducing kernel Hilbert space. It is shown that under certain conditions an estimate exists and is unique.

A Monte Carlo study was done to compare the proposed estimate with the two parametric estimates most commonly used. These parametric estimates are based on the assumption that the distributions involved are normal. The spline estimate performed as well as the parametric estimates in most cases where the distributions involved were normal. In the case of non-normal distributions the spline performed consistently better than the parametric estimates.

The computational algorithm developed can be used in the more general context of estimating a smooth function h, when we observe $z_i = L_i h + \varepsilon_i$, $i=1,n$, where ε_i's are independent, zero mean and finite variance random variables, L_i's are linear functionals, and the solution is known a priori to be in some closed, convex set in the Hilbert space, for example, the set of non-negative functions, or the set of monotone functions. This type of problem arises in areas such as cancer research, meteorology and computerized tomography.

We also consider the estimation of the logarithm of the likelihood ratio by a penalized likelihood method. Existence and uniqueness of an estimator under certain conditions is shown. However, a data based method to estimate the "correct" degree of smoothness of the estimator is not given.

END

FILMED

1-88

DTIC